AFRL-AFOSR-UK-TR-2012-0025

Game theoretic, multi-agent approach to network traffic monitoring

Dr. Michal Pechoucek

Czech Technical University in Prague
Agent Technology Center
Department of Computer Science and Engineering
Technicka 2
Prague 6, Czech Republic 166 27

EOARD Grant 10-3016

Report Date: January 2012

Final Report from 15 March 2010 to 14 January 2012

Air Force Research Laboratory
Air Force Office of Scientific Research
European Office of Aerospace Research and Development
Unit 4515 Box 14, APO AE 09421

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
16 January 2012	Final Report	15 March 2010 – 14 January 2012

4. TITLE AND SUBTITLE

Game theoretic, multi-agent approach to network traffic monitoring

5a. CONTRACT NUMBER

FA8655-10-1-3016

5b. GRANT NUMBER

Grant 10-3016

5c. PROGRAM ELEMENT NUMBER

61102F

6. AUTHOR(S)

Dr. Michal Pechoucek

5d. PROJECT NUMBER

5d. TASK NUMBER

5e. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Czech Technical University in Prague
Agent Technology Center
Department of Computer Science and Engineering
Technicka 2
Prague 6, Czech Republic 166 27

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
Unit 4515 BOX 14
APO AE 09421

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/AFOSR/RSW (EOARD)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

AFRL-AFOSR-UK-TR-2012-0025

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited. (approval given by local Public Affairs Office)

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The aim of the presented project was to investigate the potential of game-theoretical modeling of the attacker-defender interaction in the intrusion detection game. The key difference with respect to the past work in the domain is the integration of the game-theoretical methods with an actual, industry-strength anomaly detection system and operating this combination on live traffic data. Given the results presented in Section 3.8 and elsewhere in the paper, we can conclude that the simplified model presented in Section 3.2 and the extensive game model used in the core of this work (defined in Section 3.3 and solved throughout the rest of the Chapter 3) can be integrated with live IDS under reasonable, but very restrictive assumptions. In practice, we conclude that this area of research has a very high potential to produce relevant, deployable results within next 5 years, based on the trend in the computational power of current processors, average memory required for the computation and the growing sophistication of methods for efficient solving of realistic IDS games.

The critical assumptions that need to be addressed are related to the following problems: 1) Time representation and time-related assumptions in the game theoretical model. 2) Handling of several concurrent attackers. 3) Identification of attacker coalitions, where the actions from several attackers aim to achieve a common goal. 4) Sufficient detection precision and more thorough, two-way integration between the IDS and the game-theoretical model.

We argue that further, carefully designed applied research in this area should concentrate on progressive reduction of assumptions that are currently necessary to make the game theoretical model computationally solvable. The advances in game theory and gameplaying will need to be reflected in the security games domain. Further IDS and network security research is necessary to address the assumptions from the other side: by identifying the coalitions of attackers and treating them as a single individual (through the analysis of peer-to-peer networks), through more precise classification of events and through advances in the detection sensitivity and precision.

In the same time, the attackers are improving the attack techniques on their side as well, therefore ensuring that the security of future critical networks will remain an ongoing struggle.

15. SUBJECT TERMS

EOARD, game theoretic, network traffic monitoring, information security, network security, multi agent systems

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18, NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			JAMES H. LAWTON, Ph.D.
UNCLAS	UNCLAS	UNCLAS	SAR	134	19b. TELEPHONE NUMBER *(Include area code)* +44 (0)1895 616187

Game theoretic, multi-agent approach to network traffic monitoring

Final Report

This **Final Report** constitutes the Deliverable 2 of the project FA8655-10-1-3016.

Michal Pěchouček[PI], Karel Bartoš, Branislav Bošanský, Martin Grill, Martin Komoň, Viliam Lisý, Tomáš Pevný, Radek Píbil , Martin Rehák, Jan Stiborek and Michal Svoboda

Agent Technology Center, Department of Computer Science and Engineering,
Czech Technical University in Prague

Abstract: This report presents the results achieved in project "Game theoretic, multi-agent approach to network traffic monitoring." funded under the award FA8655-10-1-3016. It is referred to as ITEM 0004 in the contract documentation.

The project aims to use the advanced techniques from the field of game theory, extensive games, planning and related AI domains to improve the results of Network Intrusion Detection System (IDS). The goal of game theory use is to enable the IDS to use its past observations of possible attacker's activity in order to predict possible attacker's actions and to reconfigure itself in order detect these actions with higher probability. The studied system is based on the Network Behavior Analysis design: it processes only the network statistics and avoids the explicit analysis of the transferred data content performed by more traditional IDS systems. As such, it complements those systems by covering the network against the threats inside the perimeter, where the potential attackers would be highly professional, well informed and would target specific high-value targets. The long-term goal of our research is to be able to detect the relevant actions, infer the intentions of the attackers and reduce the likelihood of their success.

Contents

Chapter 1

Introduction

1.1 Introduction and Statement of Work

The project "Game theoretic, multi-agent approach to network traffic monitoring", also referred to informally as GAMNEP, aims to improve the current generation of anomaly-based Intrusion Detection Systems by allowing them to reason about the opponent, its intentions and its plans. The project addresses the full scope of the problem: from the reception of raw traffic statistics in the NetFlow format through automatic construction of game trees and identification of the optimal strategy, which is then adopted by the system. In order to accomplish this task, we base ourselves on the results of the past projects in the Intrusion Detection Field, financed by US ARMY CERDEC (through ITC-Atlantic) and the projects in adversarial reasoning funded by AFRL/EOARD.

Network Intrusion Detection/Protection Systems (IDS) are designed to identify and possibly block undesirable traffic detected on computer networks. Most of the current systems [49] are based on the signature matching paradigm: they inspect the content of the transferred data and look for the signatures of known attacks, in principal being similar to the anti-virus solutions. While being very effective against well-known threats, these systems are ineffective against new kinds of attacks: they are not able to combat threats that cannot be defined by a simple signature, e.g. polymorphic malware, custom-written malware/attacks, malware command & control traffic or information exfiltration. Therefore, organizations that need an additional level of security against advanced attackers typically protect themselves with the IDS systems based on the anomaly detection principle [19]. These systems do not try to identify specific attacks, but rather use past behavior of the network to predict its future behavior and report any significant differences between the prediction and the actual behavior. The system described in this report is network based and uses the NetFlow data as its input. This characteristics, together with the anomaly-detection paradigm classifies the system as a **Network Behavior Analysis** solution (NBA) in the NIST nomenclature [49].

As the network conditions change over time, the IDS may need to change its behavior/configuration to **adapt** to the changing conditions. Formal approaches based on decision theory [54] and control theory [27] are commonly used to formalize the adaptation process and to ensure that basic properties are satisfied in wide range of naturally occurring environments. The key parameters that we seek to influence by the adaptation of any anomaly-based IDS are its *sensitivity* (detection rate or true positive rate) and the *false positive* rate. True Positives (TP) are the malicious events correctly detected by the system, while the False Positives (FP) are the false alarms raised

by the system. Any practically usable IDS needs to keep the rate of false positives down, while maintaining the sensitivity.

In this project, we don't aim to perform blind adaptation based on decision-theoretical approaches, but we use our knowledge of opponents techniques, tactics and possible intentions to perform selective adaptation based on system's observation of past opponent's actions. Furthermore, we need to be aware of the possibility that the opponent is targeting the detection capabilities of our system and attempts to avoid being detected. In this Technical report, we present the following Sections, each of them concentrating on one problem aspect:

- Section 1.2 presents the system architecture, legacy self-monitoring, adaptation processes, and briefly introduces the basics of the CAMNEP IDS. We don't fully include all details of the presented components, but rather refer the reader to published results for more details about specific processes. This section does not correspond to a specific Research Target (RT), except the target **RT5**, which is dedicated to *Prototyping and Empirical Analysis*.
- Section 2.1 presents the techniques and tools used to model the hypothetical attacker's action within the system with higher sensitivity. These actions are not performed on the network, but we rely on the concept of Challenges (Section 1.2.4) to insert them into the system in order to securely evaluate its response to baseline attack techniques. The results of this process are then used to construct the game-theoretic models in Chapter 3. This section presents a contribution to **RT2**, dedicated to the simulation component of the *Opponent modeling* task.
- In Section 2.2, we present the approaches used to perform the event extraction, action recognition, and opponent modeling. These components are used to extract meaningful, symbolically described actions from the observed anomalous traffic, build behavior profiles of network resources and use the profiles and identified actions both for system output and for extensive game modeling introduced in Section 3.3. This section also presents a contribution to **RT2**, dedicated to the behavior profile identification and action recognition component of the *Opponent modeling* task.
- In Section 3.1, we present an updated version of the formalism describing the interaction between the defender and the attacker inside an IDS as a sequence of single-stage games with no transfer of information between the stages. This model is used as a formalism for the regret minimization adaptation algorithm presented in Appendix B. Section 3.1 directly contributes to the **RT3**, *Attacker/defender interaction model*. The research presented in Appendix B partially contributes to **RT4**: *Runtime IDS/P strategy selection algorithm* as a first baseline contribution in this direction.
- In Section 3.3, we present a far more elaborate formalism for the description of attacker-defender interaction. This mechanism, based on games in extensive form, allows us to model the actions of the adversary (detected by the mechanism from Section 2.2), the adaptation decision taken by the system, and the effect of both decisions on system performance. Through self-monitoring by means of Challenge insertion allows the model to reason also about the actions which were not detected by the system, possibly as a result of adversarial manipulation. This section is a core contribution to **RT3**, as it elaborates on the simple model defined in Section 3.1 and enriches the model with extensive-game formalism which allows us to use more elaborate reasoning strategies.
- In Section A, we address the problem of adversary's manipulation of system's detection algorithms. Though still hypothetical and extremely difficult to elaborate against a dynamic ensemble-based system, we still show that individual anomaly detection algorithms can be

easily avoided by a skilled attacker. This section presents a contribution towards the **RT1**: *Domain Analysis*, as it studies the deception actions unique to the IDS game.

1.1.1 Publication Results

The project has resulted in five accepted publications:

- **SSCI 2011**: Martin Rehák, Jan Stiborek, Martin Grill: *Intelligence, not Integration:Distributed Regret Minimization for IDS Control*. In Computational Intelligence in Cyber Security (CICS), 2011 IEEE Symposium [CD-ROM]. Piscataway: IEEE, 2011, p. 217-224. ISBN 978-1-4244-9905-2.
- **AAMAS 2011**: Martin Rehák, Michal Pěchouček, Martin Grill, Jan Stiborek, Karel Bartoš: *Game Theoretical Adaptation Model for Intrusion Detection System*, to appear in Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems - Innovative Applications Track (AAMAS 2011), Tumer, Yolum, Sonenberg and Stone (eds.), May, 2-6, 2011, Taipei, Taiwan.
- **IAT 2011**: Martin Rehák, Jan Stiborek and Martin Grill: *On the Value of Coordination in Distributed Self-Adaptation of Intrusion Detection System*. In Proceedings Web Intelligence and Intelligent Agent Technology WI-IAT11. Los Alamitos, CA: IEEE Computer Soc., 2011. ISBN 978-0-7695-4513-4.
- **IWCMC 2011**: Karel Bartoš, Martin Rehákand Vojtech Krmíček: *Optimizing Flow Sampling for Network Anomaly Detection*, to appear in Proceedings of the TRaffic Analysis and Classification (TRAC) Workshop (IWCMC2011-TRAC), July 2011, Istanbul, Turkey
- **SPI 2011**: Martin Rehák, Karel Bartoš, Martin Grill, Pavel Jisl, Jan Jusko, Tomas Pevný, Michal Svoboda a Jan Stiborek: *Strategic Self-Organization Methods for Intrusion Detection Systems*, to appear in Proceedings of Security and Protection of Information, May 2011, University of Defence, Brno

Two of the accepted publications appear at major international conferences, the third is presented at highly-specialized traffic analysis workshop before submission to more selective venue and the last is a presentation for the Army technologist and security researches in the region.

1.2 High-Level Design

This section provides a quick introduction into the problem area and presents the architecture and components of the underlying Intrusion Detection Software. It does not aim to present the innovative research performed during the project, but presents the system architecture and relationships between the components, so that the reader can understood the implications of domain specifics on game formulation.

1.2.1 System Concept

The Network Behavior Analysis [49] algorithms are primarily designed to identify strategic, persistent threats operating inside the protected perimeter after having evaded the detection and protection deployed on perimeter gateways. Unlike most Network Intrusion Detection Devices or security appliances, they don't analyze the content of the transmitted information, but use the statistics (Fig. 1.1) in the NetFlow/IPFIX format [15, 14] to build, maintain and combine behavior models. The behavior models capture the normal behavior of individual components (hosts or services), relationships between the activity of the hosts and other traffic features. These models, based on the past behavior of the monitored network, are then applied to the observed traffic and the possible discrepancies between the model prediction and actual observation - *anomalies* - are reported as possibly resulting from a malicious action.

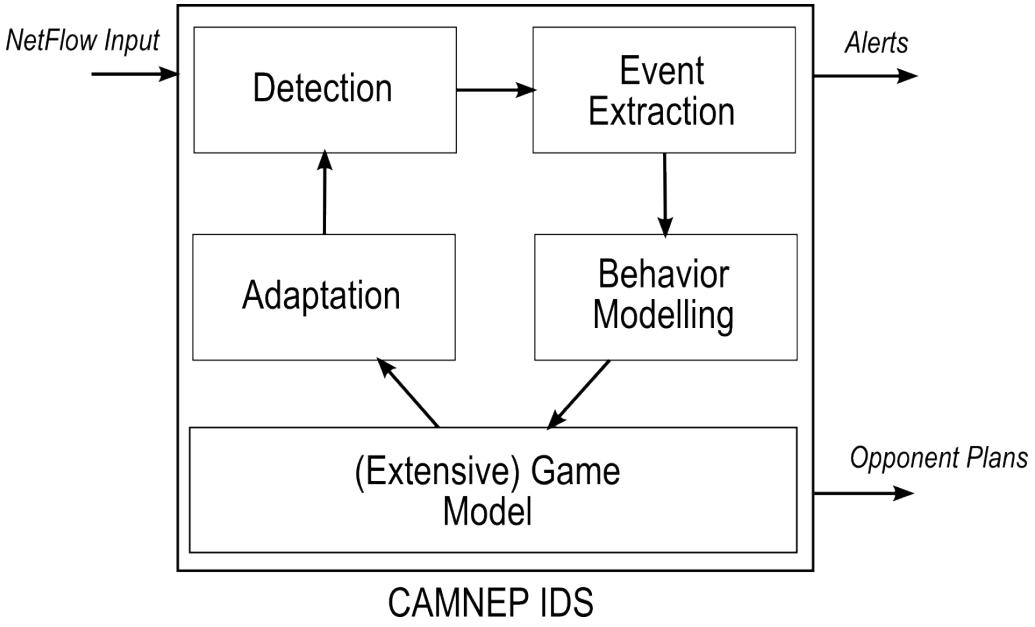


Figure 1.1: Main Functional Blocks Interacting in GAMNEP project.

In Fig. 1.1, we show a highly simplified diagram of the CAMNEP system in the version designed and maintained as an experimental testbed for the GAMNEP project. The system receives the NetFlow data from network components and processes them in the Detection component. The Detection component presents its output to the Event Extraction subsystem, which interacts with Behavior Modeling to create concise, consistent, and classified set of events based not only on current behavior, but also on the past behavior of the resource associated with the Event. The Events are then accessible to system users, but in the same time, the same events processed

through the behavior modeling, are used as recognized Actions for the construction of the game-theoretic model of the interaction between the attacker and the system. The results of this model's processing are then used for the Adaptation of the Detection layer, so that the system would be better optimized to detect the actions typically associated with the expected attacker's behavior.

Duration	*Proto*	*Src IP Addr:Port*	*Dst IP Addr:Port*	*Flags*	*Pack.*	*Bytes*
0.400	TCP	192.168.195.164:1086	192.168.10.12:445	.A....	2	84
0.000	TCP	62.97.162.208:3417	192.168.192.83:1172	.AP...	1	42
0.577	TCP	192.168.195.132:2544	194.228.32.3:80	.A.R..	3	126
0.576	TCP	192.168.195.132:2545	194.228.32.3:80	.A.R..	3	126
0.000	UDP	192.168.60.31:4021	192.168.19.247:53	.....	1	55
0.000	ICMP	192.168.19.247:0	192.168.60.31:0	.....	1	149
0.000	UDP	192.168.60.31:4021	192.168.60.1:53	.....	1	55
0.000	UDP	192.168.60.31:4020	192.43.244.18:123	.....	1	72
30.276	TCP	192.168.192.170:61158	71.33.170.53:1358	.AP...	307	368627
0.000	UDP	24.28.89.160:63319	192.168.192.83:58359	.....	1	42
0.000	TCP	63.208.197.21:443	192.168.192.106:1031	.AP...	1	73
0.093	TCP	192.168.193.58:1302	192.168.192.5:110	.AP.SF	8	356
0.093	TCP	192.168.192.5:110	192.168.193.58:1302	.AP.SF	8	440
0.000	UDP	85.160.81.10:6766	192.168.192.217:11084	.....	1	45
0.000	UDP	192.168.192.217:11084	85.160.81.10:6766	.....	1	45
0.000	TCP	192.168.19.247:1723	192.168.60.19:1042	.AP...	1	56

Table 1.1: An example of NetFlow data (partially anonymized and with time information removed) as received from NFDUMP tool [26]. Supplementary derived information, such as packets per second, average bytes per packet, etc. is not represented in this table. One line corresponds to one network flow, typical file acquired over 5 minutes on gigabit network can have 1-2 millions of lines.

The high-level view above is further technically enhanced in the next section, where the software components are presented in higher level of detail, and will be further extended throughout the report.

1.2.2 Architecture Overview

In this section, we present the high-level overview of the complete system at the level of a single node, before getting into a mode detailed description of system components and the principles behind its operation. In Fig. 1.2, we can see that each individual node consists of several subsystems interacting with each other. Most of the subsystems are implemented as one or more **agents**, and all agents communicate solely by the exchange of messages. The whole system is written in Java and is based on the AGLOBE multi-agent platform [56]. The use of multi-agent paradigm ensures high robustness and resilience to faults, while the use of AGLOBE ensures high efficiency of processing and low performance overhead of multi-agent computation.

The functional components of the system are:

- **Traffic acquisition layer** responsible for reception and preprocessing of NetFlow/IPFIX data;
- **Detection layer** responsible for identification of anomalous flows;
- Challenge-based **self-monitoring**, which is used to evaluate the performance of the detection layer in the real-time;
- **Game-Theoretic Adaptation** process, which uses the results of self-monitoring to optimally adapt the system under given circumstances;

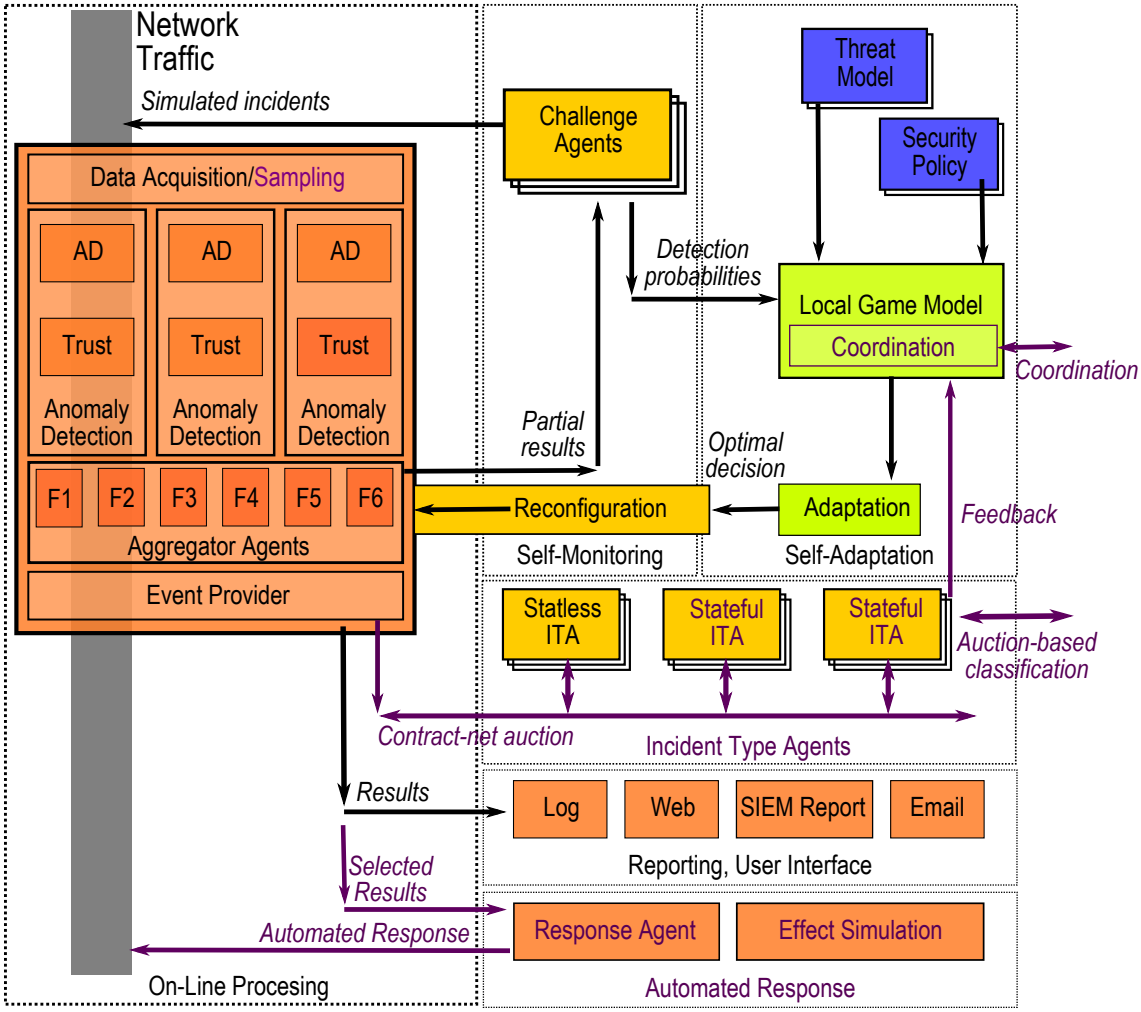


Figure 1.2: Architecture overview of the standard configuration of the CAMNEP system.

- **Event Extraction** and **Behavior Modeling** is a set of agents that converts the suspicious flows into coherent events and classifies them;
- **Results Exportation** and **Integration** is a set of component that ensure that the results of system operation are accessible for other systems and the end-user;
- **Automated Response** Components (out of scope of this report) that will use the system output to prepare the prevention rules for active response to the attacker's actions;
- **Web-Based Visualization** (independent), which displays the system results in a standard web-based interface.

In the following, we will briefly introduce the components and layers of the CAMNEP system, in order to give the reader a level of understanding sufficient for the comprehension of the effects that can affect the results of the interaction between the attacker and the IDS (defender). Our implementation is rather unique by its tight integration between the real-world software and fairly abstract game-theoretic approaches deployed in the online mode, but this also necessitates deeper knowledge of the techniques involved.

1.2.3 Traffic Acquisition Layer

The components in this layer are designed to receive the NetFlow data from one or more sources over a pre-determined period of time. The Traffic Acquisition functionality is fully implemented inside the `NetFeeder` agent. This (rather large) agent currently includes the following functional modules, presented in order of invocation:

- **NetFlow collection**. In the first stage of processing, the NetFlow sources streaming into the system have to be identified. For each source, the system needs to evaluate the structure of NetFlow information provided by the NetFlow source – most often using the Templates periodically inserted into the UDP stream of NetFlow data. In the case of NetFlow v5, the processing of the data begins immediately after the reception of first valid NetFlow record. In the case of NetFlow v9 or IPFIX, the processing starts once the first template record has been received. The template is required for the transformation of NetFlow into the standard storage structure used inside the system.

  The NetFlow data streams from all sources are directed into a single UDP port (port 3000 is the default value). The system considers the flows received from all sources as equivalent and does not perform any selection, removal of duplicates or sorting based on the source information. Liveliness of the source is not verified either – if one of the sources fail to provide the data in a given time period, this failure is neither reported nor considered in the subsequent computation.

- **DataSet preparation** and timing. In the second stage of processing, the system gathers the data received during one observation period (typically 5 minutes long) and stores them in a unified data structure. The flows to be stored are determined as the flows received from all valid sources during the 5-minute window, based on the local clock of the CAMNEP system. On the other hand, the timestamps included in the NetFlow records are not transformed, and may fall outside of the particular 5-minute window for a multitude of reasons, including clock synchronization, buffer timeouts of NetFlow probes and time zone issues. However, this rarely affects the quality of processing and the timing model based on the local clock was selected as the most robust for distributed environments. This change presents the

departure from the past practice, when the CAMNEP core accepted the dataset assembled and collected by third-party collector software.

The dataset delimitation in time effectively defines one stage of the adversarial plan recognition game as introduced in Section 3.3 (It also defines the successive single-stage games when using the alternative problem formulation).

- **Challenge insertion implementation**. The flows received from the probes are enriched with the simulated flows inserted by the Challenge (or Incident) agents [44] introduced in Section 1.2.4.
- **Secondary feature computation** and storage. Once all flows (both acquired from the network and simulated) are stored, the system determines the secondary features on the significant flow subsets, most notably the aggregates and statistical moments for individual traffic sources (srcIP addresses), individual destinations and other subsets. These features are subsequently stored in a dedicated structure associated with the dataset containing the flows and specific features are used to enrich the features of the individual flows.
- Optional **sampling** of input data. Sampling is employed by the system to handle the situations when the volume of the monitored traffic exceeds the capacity of the hardware configuration on which the system runs. The system is able to automatically select (see Appendix B) from several sampling strategies (also described in paper in Appendix B).
- **Detection process invocation** and liveliness checks are the final responsibility of the NetFeeder agent. The dataset produced by this agent is sent to all registered detection agents and the NetFeeder follows its processing in order to possibly delay the sending of the next dataset should the processing of the previous one had not been finished yet.

1.2.4 Detection Layer Features

The detection layer is the key layer of the system, as it performs the primary identification of the anomalous flows. It receives the flows assembled by the NetFeeder agent into the dataset and assigns them the aggregate trustfulness value, according to the opinion of each detection algorithm, i.e. each flow receives one trustfulness value which places the flow on the scale from normal, 1, to fully anomalous, 0.

Detection Layer Overview

The Detection layer consists of **Detection Agents**. These agents are the key elements of the detection process, as they assign the anomaly/trustfulness values to individual flows. Each of them is based on one particular anomaly detection method, typically based on existing research in the network anomaly detection. They (currently) operate in two stages:

- **Anomaly detection**: Each detection agent is based on one anomaly detection method, either designed in-house, or based on past published work. The anomaly detection methods use their model (internal state) and the input - flows in the current dataset - to determine an immediate anomaly of flows in the dataset. The anomaly detection methods are based on the following approaches and types of features:
  - ⋆ entropies of flow characteristics for individual source IP addresses [58],
  - ⋆ entropies of flow characteristics for individual destination IP addresses [59],

  - ⋆ deviation of flow entropies from the PCA-based prediction model of individual sources [32],
  - ⋆ deviation of traffic volumes from the PCA-based prediction for individual major sources [31],
  - ⋆ rapid surges in the number of flows with given characteristics from the individual sources [20],
  - ⋆ ratios between the number of destination addresses and port numbers for individual sources [53],

  The anomaly values are provided by each agent for all flows in the dataset and are exchanged between the Detection Agents, so that they can be combined and used as an input for the update of trust models maintained by each agent.

- **Trust modeling**: The purpose of the trust modeling [42] as deployed in CAMNEP [43] is to build a higher quality assessment of the individual flow anomaly, based on the opinions built in the trust model over time from the anomaly values submitted by all detection agents. All trust models process identical inputs: the flows from the processed dataset and the anomaly values submitted by all detection agents. The results are differentiated by the features used to construct the trust model, built as a set of clusters in the space where the flows are represented, each cluster centered on one *centroid*. The trust model of each agent clusters the traffic according to the characteristics used by the anomaly detection model of the agent (similarity between the flows). This results in a different composition of clusters between the agents, as any two flows that may be similar according to the characteristics used by one agent can be very different to other agent's models. Once the agents build the characteristic profiles of behavior, they use the anomaly values provided by the anomaly detection methods of all agents to progressively determine the appropriate level of trustfulness of each cluster. This value, accumulated over time, is then used as an individual agent's assessment of each flow.

The trust-based algorithm improves the quality of the classification by parallel trust modeling performed by the detection agents and the adaptive integration of their outputs into the final trustfulness assessment. The trust modeling inputs are identical for all agents, and the principal improvement comes from the fact that each of the agents uses different traffic features to define its feature space and its metrics. Due to these differences, each trust model uses its own criteria to perform the aggregation of data. The features of each such model are derived from the features that separate the malicious traffic from the legitimate traffic (because they are defined by an existing anomaly detection method). Therefore, the attack flows as identified by any individual agent are typically concentrated in a rather limited part of the feature space, covered by few centroids. If a group of flows (such as the flows from the single source) falls into the untrusted region in a feature space of all agents, the trustfulness of the centroids in this region will be consistently low, because it was accumulated from highly anomalous traffic (Fig. 1.3 left). On the other hand, if only one or few of the agents identify the traffic as malicious, the flows will likely to be dispersed over many centroids in the trust models of other agents and will have higher trustfulness (Fig. 1.3 right). This hypothesis, which explains how does the trust modeling stage improve the false positives rate, was verified empirically during our experiments: when an agent identifies a particular event as a false positive, the flows that constitute the event are spread over a relatively high number (more than 20) of centroid-defined clusters, while the attack flows are normally associated with less than 5 centroids.

Trustfulness Aggregation.

The Detection agents provide their trustfulness assessment (conceptually a reputation opinion) to the aggregation agents. There are several types of the Aggregation agents, created during the past

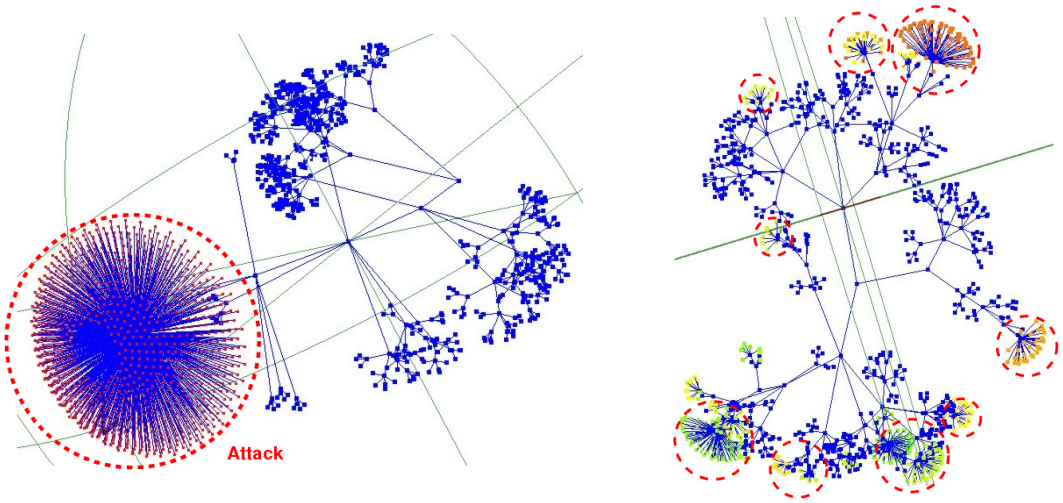


Figure 1.3: A peek into the trust model of a detection agent. The flows are displayed as tree extremities attached to the closest centroid of the trust model. Note that the attack flows are concentrated next to a single centroid (left), while the normal traffic from legitimate source is spread over the whole model (right).

projects and improved during the current project:

- The **Master Aggregator** agent is a highly simplified component that directs the aggregation process and provides a fallback solution should the dynamic aggregation operator selection fail or fail to finish in time. In that case, it performs a simple arithmetic average of trust values submitted by individual Detection agents and this result is used as a detection layer output.
- One or more **Dynamic Aggregator** agents contain one or more aggregation operators. These operators, based on Order-Weighted Averaging [11, 60], identity-weighted averaging or a combination of both constitutes one possible system output by combining the detection agent's trustfulness opinions. The identity-weighted averaging is a fairly straightforward operation, as it assigns the weights to the trustfulness provided by individual agents in the system and averages the trustfulness values according to these weights. The Order Weighted Averaging (OWA) is different, as the weights are not assigned to the individual agents, but to the order of trustfulness values provided by the individual detection agents. The mechanism can thus emphasize the lowest or highest trustfulness values or can only use the values from the middle of the distribution.

The Master Aggregator agent receives the results of aggregation for the flows from the Dynamic Aggregator agents and uses the game theoretical modeling methods described in this report to select the aggregation strategy with optimal sensitivity against expected next attacker's action.

Challenge Insertion

In order to resolve the question which of the aggregation operators prevails, we use the self-adaptation infrastructure based on the Challenge insertion [54, 44]. We use the term *Incident Provider* to denote a dedicated agent that controls the system adaptation by implementing the

mechanism described in this paper. The Incident Provider creates individual *Challenge agents*, each of them representing a specific incident in the past, and these temporary, single purpose agents interact with the data-provisioning layers of the system in order to insert the flows relative to the incident into the background traffic and to retrieve and analyze the detection results provided by the aggregation operators. The results are then reported to the user agent and used to select the optimal aggregation operator's result as a system output for the current dataset. Challenge processor also dynamically determines the number of Challenges to insert into the next dataset, as we have described earlier [46, 45].

The self-adaptation process (detailed in Fig. 1.2) is based on the insertion of challenges into the background of network flow data observed by the system. The challenges are represented as sets of NetFlow records, corresponding to classified incidents observed in the past. These records generated by individual challenge agents are mixed with the background traffic, so that they cannot be identified by the detection/aggregation agents. They are processed together with the rest of the traffic, used to update the anomaly detection mechanism data and trust models of individual detection agents and are evaluated with the rest of the traffic. Once the processing is completed, they are re-identified by their respective challenge agents, removed from the user output and the anomaly attributed to these flows by individual aggregation agents is used to evaluate these agents and to select the optimal output agent for the current network conditions.

There are two types of challenges. The *malicious challenges* correspond to known attack types and the *legitimate challenges* represent known instances of legitimate events that tend to be misclassified as anomalous. We further divide malicious challenges into classes according to the type of the attack, such as fingerprinting/vertical scan, horizontal scan, brute force password cracking, etc. With respect to each of these attack classes, we characterize each aggregation agent by a probability distribution, empirically estimated from the continuous anomaly values attributed to the challenges from this class, as we can see in Fig. 1.4. We also define a single additional distribution for all legitimate challenges.

System response to challenges drawn from various attack classes or representing the legitimate behavior allows to estimate the reaction of the system to the modeled attacks. Specifically, the challenges are used to establish detection rates for each attack class and the false positive rate for the system as a whole. These crucial values are then used in the dynamic adaptation process described in the next section.

It shall be noted that the self-adaptive features of challenge insertion, which determine the number of challenges to insert into each specific dataset **do not depend** on the current adaptation method used, but are based on the trust-based formalism introduced in [46, 45]. This allows the system to directly compare the results of various adaptation methods without being concerned with the meta-effects of the adaptation.

In Section 2.1, we present a novel approach to Challenge creation, based on the combination of approaches from automated planning with more traditional security techniques. This allows us to reach higher fidelity in system models and to evaluate the system capabilities more realistically.

1.2.5 Dynamic Selection of Detection (Aggregation) Strategy

In this section, we are going to describe the existing approaches to strategy selection in the CAMNEP system.

– **Trust-Based** aggregation operator selection (Fig. 1.5), presented in [45] uses the FIRE trust model applied to the results of challenge insertion to identify the operator that achieves the

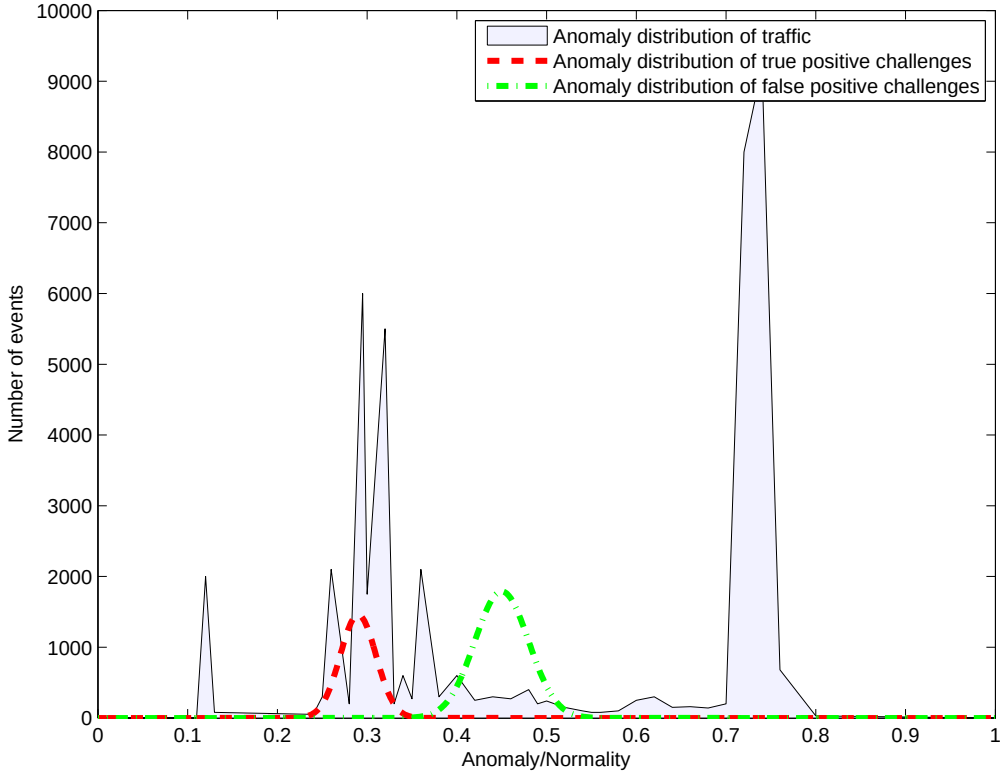


Figure 1.4: Distribution of challenges on the background of the anomalies attributed to the traffic from one traffic observation interval. The distribution of anomaly of the malicious challenges (from one class) is on the left side of the graph, while the legitimate events are on the right.

best separation of malicious and legitimate challenges. For each aggregation operator, we maintain a trust model based on the individual trust dimension of the FIRE model [41], which is set up to emphasize the last 5 datasets. On each dataset, we determine the normalized distance between the average trustfulness attributed to the legitimate challenges (borderline cases representing typical false positives) and the average trustfulness attributed to the malicious challenges (separated into attack classes) and the normalized distance is used as a trust observation for the FIRE individual trust model of the respective aggregation operator. As mentioned earlier, the trust-based challenge evaluation is also used to regulate the number of challenges inserted into the network traffic, regardless of the method used for the adaptation.

– **Single-Stage Game** based selection using the **Max-Min** strategy selection algorithm (Fig. 1.6). Unlike in the previous case, where we only consider the problem from the defender's perspective, in the Max-Min case we analyze the possible strategies of the opponent and their likely impact on the defender's utility. Then, CAMNEP, playing the role of the defender, determines which aggregation strategy to adopt by identifying the worst possible outcome of each defender's strategy (Min) and then selecting the strategy which maximizes the worst outcome (Max-Min). This approach is well suited for the situations where the opponent's gain is equal to the defender's loss, i.e. zero-sum games [36], but it doesn't use any available information about the opponent's intent or immediate payoffs. The utility function of the defender is constructed using the inputs from the challenge-based self-monitoring. The game model use for adaptation is described in Section 3.2.

– **Single-Stage Game** based selection playing the **Nash Equilibria** strategy (Fig. 1.7). The

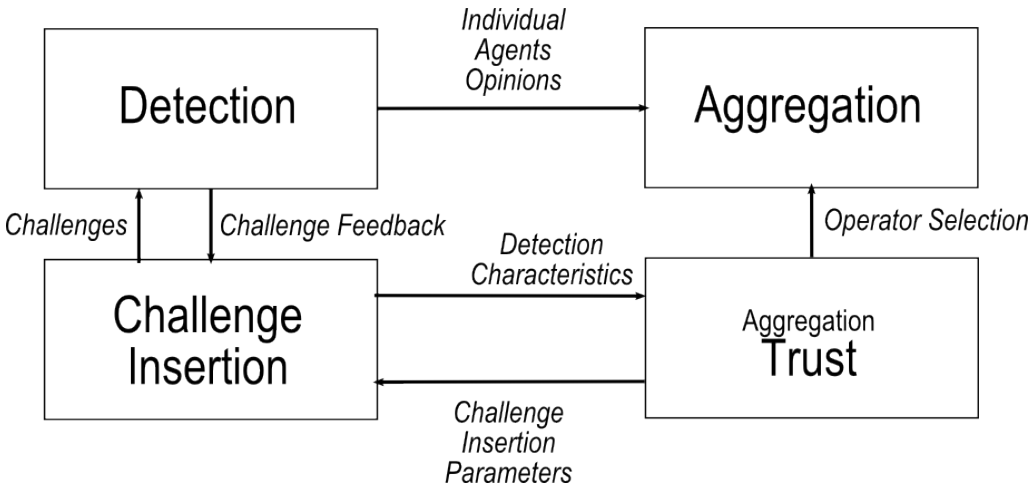


Figure 1.5: Trust-Based Adaptation Overview.

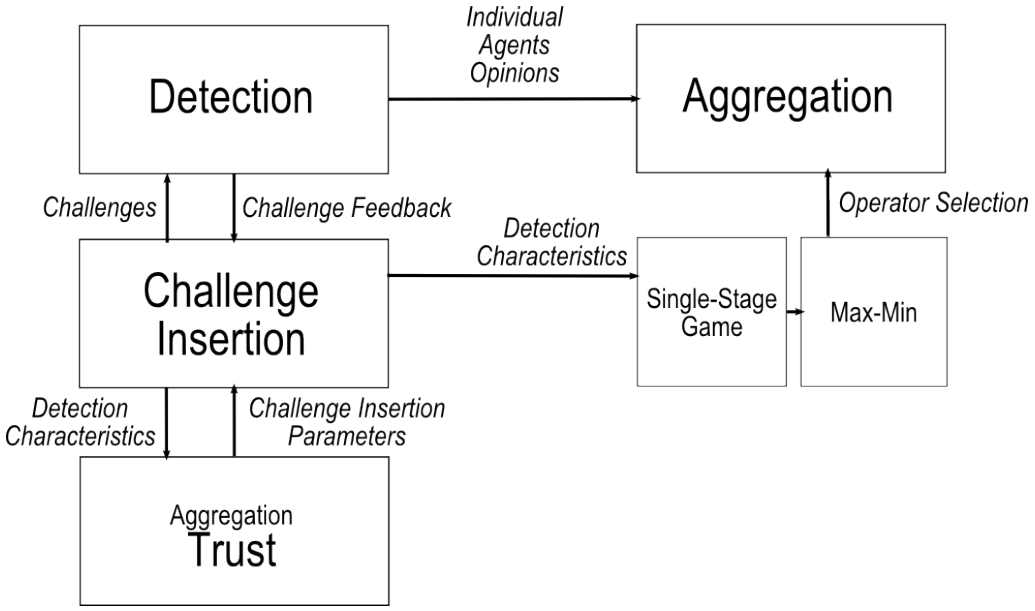


Figure 1.6: Adaptation based on a single-stage game solved by the concept of Max-Min

underlying interaction model is the same as in the previous case, but the use of Nash equilibria as a strategy requires the defender (CAMNEP) to reason about the utility function of the opponent. It also requires the opponent to reason about the internal state and the utility function of the IDS. Playing Nash equilibria should deliver slightly better results than the Max-Min approach, as it assumes that the attacker is rational and some of the possible "worst" strategies can be eliminated from consideration. However, this mostly applies on larger spaces, and the experimental results in Section 3.2 suggest that both methods result in similar payoffs for the defender. Compared to direct trust-based aggregation selection, the game-theoretic strategies result in safer selection of the aggregation strategy (operator) which is more difficult to predict by the opponent.

– **Global Regret Minimization**, presented in detail in Appendix B of this report, uses a

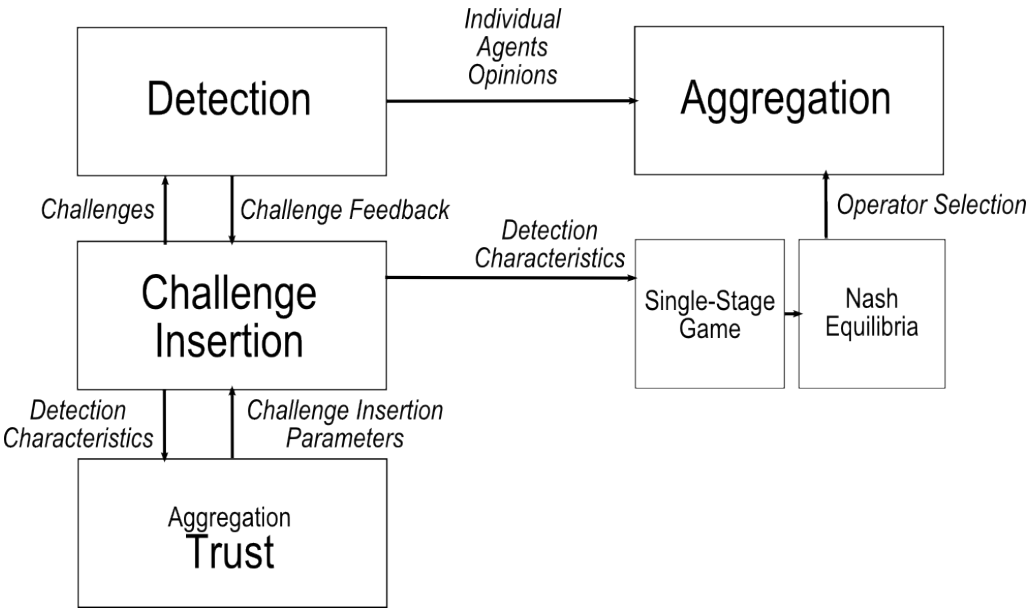


Figure 1.7: Adaptation based on a single-stage game solved by the concept of Nash Equilibria

more elaborate model of the attacker-defender interaction. The game is structured as a sequence of identical two-player games, but the choice of the strategy is performed using external regret minimization. In the Global case, we optimize two parameters at once, with full synchronization and considering them as a single strategy drawn from the Cartesian product of two sub-strategies: (*i*) sampling strategy selection and (*ii*) aggregation operation selection. This scenario has been implemented for research and experimentation purposes only – as we use the external regret minimization in both cases, the system needs to perform all sampling strategies at once and to evaluate the response of all aggregation operation to sampled challenges in all sampling variants, making such installation unappealing for obvious efficiency reasons. This problem is addressed by the alternative method described below.

– **Concurrent, Two Layer Regret Minimization**, which is also presented in Appendix B uses an alternative formulation of the optimization problem, which imposes less constraints on system components. The selection of two strategies is performed independently, by two separate processes running in two distinct layers with no explicit coordination regarding the selected strategy. Obviously, in this setup, the selection of sampling strategy significantly affects the performance of individual detection algorithms and subsequently also the optimality of aggregation strategies.

The baseline approaches presented here will be complemented by the approach based on adversarial plan recognition game, which will be introduced in Section 3.3.

1.2.6 Event Creation and Characterization

The Detection layer processes the dataset provided by the Data Acquisition layer and its processing, including the adaptation step, results in a trustfulness assignment which divides the traffic into the normal and anomalous classes of flows. The flows are then processed by the *Event Provider* agent, the core of the Event Creation and Characterization components of the system.

Event Provider cooperates with the Incident Type Agents to transform the anomalous flows into coherent events with appropriate symbolic description. The process of Event Creation and classification is not unidirectional, but is based on several stages involving event nucleus creation, pre-classification, event fusion and final event classification. The details of this process are presented in Section 2.2.

Chapter 2

Modelling Attacker and Defender in CAMNEP

2.1 Attacker Simulation

As we have described above, the CAMNEP system is able to adapt to the network situation using the challenge insertion process. This process influences the selection of aggregation operator as well as the threshold dividing malicious and legitimate flows. Therefore, the challenge insertion process is crucial for the system and affects performance of the whole intrusion detection process. The inserted challenge flows are created from the captures that are selected randomly from the database, according to their type and security policy defined by the administrator. This implies that selected challenges are independent of each other and to the current network situation. With respect to these properties, challenges can be used well to simulate attacks on the monitored network.

Although the challenge insertion mechanism described so far significantly improves the adaptation capability of CAMNEP, it is limited in several ways. First, it cannot simulate persistent or strategically behaving attackers following pre-defined attack plans to achieve their goals. Second, the database of challenge flows contains captures of attacks without any background noise typical for the attackers, who use other network services during the attack. Finally, there is a need for the system to be able to simulate legitimate behavior of servers difficult for the intrusion detection systems. Simulation of such servers needs the insertion process to support insertion of long periods of constant behavior. Moreover the challenge flows used so far do not contain neither replies of the attacked machines nor side effects of the attack actions (e.g. DNS resolving during SSH bruteforce or scans).

To address all these problems we need the challenge insertion process and the challenge flows to be as realistic as possible. Hence, instead of inserting randomly selected attack flows of various types into the system's input we need more realistic simulation of the attacker.

Better challenge selection and insertion process will improve the accuracy of the system in many ways, the most important ones are:

– Using more realistic simulation of an attacker improves the estimates of detection rates of individual detection algorithms and aggregation operators.

- Improved estimates of false positive/false negative rates of different aggregation operators decreases the probability of selecting sub-optimal aggregation operator for the current configuration of the system and the state of background traffic.
- Choosing the optimal aggregation operator increases the accuracy and robustness of the CAMNEP.

In order to bring the inserted challenges closer to reality, the challenges are no longer selected randomly, but they follow realistic attack plans. The attack plans model strategically behaving opponents with realistic goals and appropriate actions leading them to the predefined goal. The attack plans (can be static or dynamic) are based on Planning Domain Definition Language [23] (PDDL) described in Section 2.1.2, which allows us to efficiently define the problem domain containing available actions (described in Section 2.1.3) of the attacker together with their preconditions and their effects, and the set of possible attacker's goals (described in Section 2.1.3) such as gaining access to some machines, distribution of malware, planting denial of service, etc.

Attack plans, which can be either static or created on-the-fly by the planner described in Section 2.1.3, enable us to simulate real attackers progressing towards their goals. These plans are also useful to simulate legitimate behavior of various problematic servers such as proxy server, DNS server, etc.

2.1.1 Architecture

Plan generation is handled by the Plan Provider agent, which is responsible for the creation of attack plans in PDDL format and for the creation of new Plan Executor agents. The philosophy behind is the same as the philosophy behind the Incident provider and the Challenge agents.

A new attack plan is initiated by the Plan Provider agent either generating new plan in PDDL format or loading static PDDL plan from the database. The choice of the static plan, or the type of generated plan, depends on the state of monitored network and number of running Plan Executor agents. After the plan is created, the Plan Provider agent creates new Plan Executor agent, which takes care of the plan execution and evaluation.

The Plan Executor agent is responsible for generating attack flows according to the plan described in PDDL language. It does so in stages, first breaking the plan into individual actions followed by generating flows implementing individual actions. The Executor agent also takes care about nifty details such as setting proper time delays between actions, adjusting IP addresses to be consistent within the plan yet non-conflicting with IPs in the monitored network, etc. The generated flows are sent to the NetFeeder agent that mixes them with the real traffic. The Plan Executor agent also observes the output of the system and gives feedback to the Dynamic Aggregator agent about performance of aggregation operators in the exactly same way as it is done by Challenge agents.

2.1.2 PDDL

PDDL (Planning Domain Definition Language) attempts to standardize planning domain and problem description languages. Although it was mainly developed for the 1998/2000 International Planning Competitions, it is nowadays supported by most available planners.

PDDL task consist of several components:

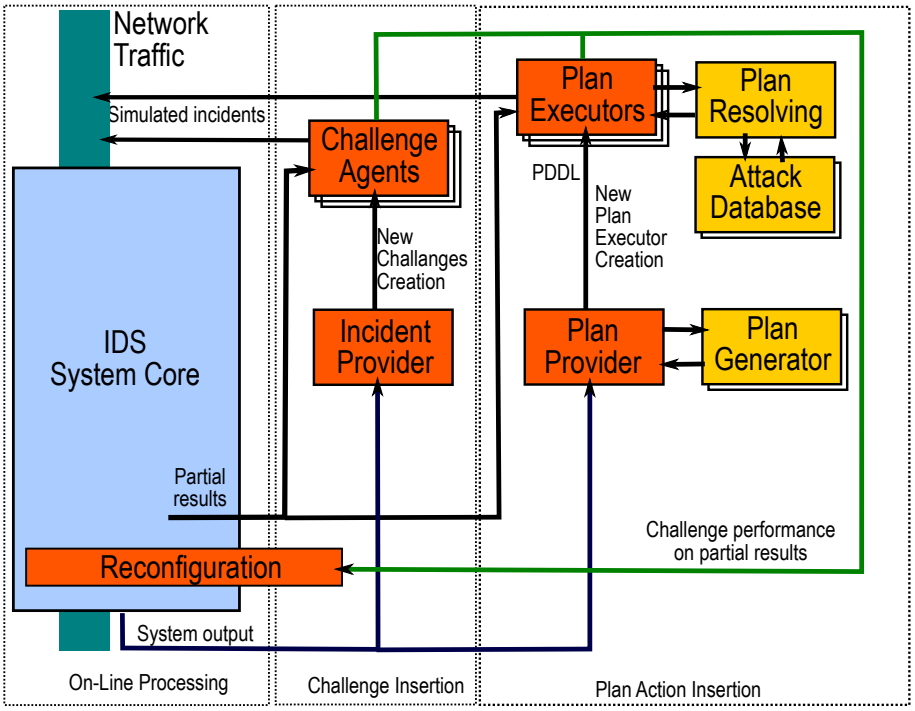


Figure 2.1: Challenge insertion architecture of both random challenge process and plan execution process.

- **Objects** are all items of interest in the world. In our case the objects are all the information that an attacker should possess or obtain during the attack.
- **Predicates** are properties of objects. In attacker planning domain the predicates specify the knowledge that attacker needs in order to perform some action (in our case an attack).
- **Initial state** is the initial state of the world, in our case it is the knowledge the attacker has before starting the attack on target network.
- **Goal specification** is the definition of the desired goal.
- **Available actions** represents actions that can change the state of the world. In our case the actions are different types of network attacks. By means of these actions, an attacker gains new knowledge about the attacked network needed to pave his road towards the goal.

The problem in the PDDL is specified by

- **Domain specification** containing all actions available in current domain together with their preconditions,
- and **Problem specification** defining the problem to be solved. This specification contains all objects, initial state of the world, and desired goal.

The usage of PDDL language provides us formalism to describe domain, action and problems which could be used for dynamic generation of attacker's plans.

2.1.3 Attacker Actions

Each PDDL action is identified by the name corresponding to the groovy script, list of parameters, preconditions, and effect of the action. For example the Action 1 describes horizontal scan parameterized by the source IP, destination IP, scanned port, and number of flows. The predicate `DUMMY_PREDICATE` means that the attacker does not need any knowledge to execute this action. When attacker executes this action, he gains the knowledge (predicate) `SUBNET_PORT_KNOWLEDGE` and he can use this knowledge to progress in his plan. More examples of attacker actions are brute force cracking of SSH password (Action 3) and upload of malware (Action 4) in Section 2.1.3.

So far we have specified more than 30 different actions in our attack planning domain. There are exploratory actions such as different types of scans, attacks to particular services like SSH bruteforcing, various rootkit usage, simulations of P2P networks and skype supernodes. Besides attacks, we created simulation of legitimate actions like web use, ftp use, dns server, proxy server etc., which are important to simulate legitimate entities on the network.

Action 1 Example of horizontal scan action

(:action **SCAN_HORIZONTAL**
:parameters (?FLOWCOUNT - INT ?SRCIP - SRCIP ?SUBNET - SUBNET ?PORT - PORT)
:precondition (DUMMY_PREDICATE)
:effect (SUBNET_PORT_KNOWLEDGE ?SUBNET ?PORT)

Action 2 Example of vertical scan action.

(:action **SCAN_VERTICAL**
:parameters (?FLOWCOUNT - INT ?SRCIP - SRCIP ?DSTIP DSTIP)
:precondition (IP_KNOWLEDGE ?DSTIP)
:effect (and(ALL_PORT_KNOWLEDGE ?DSTIP)))

Action 3 Example of SSH bruteforce attack action.

(:action **SSH_BRUTEFORCE**
:parameters (?SRCIP - SRCIP ?DSTIP - DSTIP)
:precondition (PORT_KNOWLEDGE ?DSTIP SSH_PORT)
:effect (SSH_PASSWORD_KNOWLEDGE ?DSTIP))

There are two possibilities, how to create flows from actions: either load the attack flows from the database or to generate them according to the model.

For the first approach, we have created a large database of attacks that have been isolated from observations of several networks by the CAMNEP system. Presently, we have more than 400 captures of various attacks of about 20 different types. These attack captures can be either used as stand-alone attacks, as in the original CAMNEP system, or as one action in a larger attack plan generated by the Plan Executor agent.

For the latter, we have created several models of actions performed by attackers. These models are based on observations of effects of real attackers on our university network and their influence on the network characteristics. Using these models we are able to create scripts generating flows representing attacks together with all side-effects (see an example on Figure 2.2).

Both presented approaches (loading of attack flows from database or generation of attack flows according to the model) can be used by actions defined in the PDDL. Every PDDL action has corresponding script that can either load appropriate action from the database or call attack flow-building script that creates flows that will be inserted into the CAMNEP to execute the action.

Action 4 Example of malware upload action.

(:action **MALWARE_UPLOAD**
:parameters (?SRCIP - SRCIP ?DSTIP - DSTIP)
:precondition (SSH_PASSWORD_KNOWLEDGE ?DSTIP)
:effect (MALWARE_PROPAGATION ?DSTIP))

Outgoing communication:

Duration	Protocol	Src IP	Dst IP	Src Port	Dst Port	Flags	Tos	Packets	Bytes	Flows
∼ 0	UDP	server dns	client	53	any	o o o o o o	0	1	⟨118; 328⟩	1

```
...
2010-10-22 11:38:13.684 0.000 UDP 147.32.84.131:53 -> 147.32.80.9:50593 ..... 0 1 118 1
2010-10-22 11:38:13.845 0.000 UDP 147.32.84.131:53 -> 147.32.80.9:54452 ..... 0 1 118 1
2010-10-22 11:38:14.398 0.000 UDP 147.32.84.131:53 -> 147.32.80.9:51980 ..... 0 1 118 1
2010-10-22 11:38:14.559 0.000 UDP 147.32.84.131:53 -> 147.32.80.9:56525 ..... 0 1 118 1
2010-10-22 11:38:14.718 0.000 UDP 147.32.84.131:53 -> 147.32.80.9:49533 ..... 0 1 118 1
2010-10-22 11:38:14.879 0.000 UDP 147.32.84.131:53 -> 147.32.80.9:55246 ..... 0 1 118 1
2010-10-22 11:38:15.040 0.000 UDP 147.32.84.131:53 -> 147.32.80.9:59087 ..... 0 1 118 1
2010-10-22 11:38:15.199 0.000 UDP 147.32.84.131:53 -> 147.32.80.9:57793 ..... 0 1 118 1
2010-10-22 11:38:15.359 0.000 UDP 147.32.84.131:53 -> 147.32.80.9:63011 ..... 0 1 118 1
2010-10-22 11:38:15.521 0.000 UDP 147.32.84.131:53 -> 147.32.80.9:59321 ..... 0 1 118 1
2010-10-22 11:38:15.682 0.000 UDP 147.32.84.131:53 -> 147.32.80.9:55698 ..... 0 1 118 1
2010-10-22 11:38:15.842 0.000 UDP 147.32.84.131:53 -> 147.32.80.9:60765 ..... 0 1 118 1
...
```

Incoming communication:

Duration	Protocol	Src IP	Dst IP	Src Port	Dst Port	Flags	Tos	Packets	Bytes	Flows
∼ 0	UDP	client	server	any	53	o o o o o o	0	1	⟨101; 600⟩	1

```
...
2010-10-22 11:38:16.211 0.000 UDP 147.32.80.9:51353 -> 147.32.84.131:53 ..... 0 1 101 1
2010-10-22 11:38:16.232 0.000 UDP 147.32.80.9:59351 -> 147.32.84.131:53 ..... 0 1 300 1
2010-10-22 11:38:16.250 0.000 UDP 147.32.80.9:54889 -> 147.32.84.131:53 ..... 0 1 300 1
2010-10-22 11:38:16.270 0.000 UDP 147.32.80.9:59789 -> 147.32.84.131:53 ..... 0 1 300 1
2010-10-22 11:38:16.290 0.000 UDP 147.32.80.9:60848 -> 147.32.84.131:53 ..... 0 1 300 1
2010-10-22 11:38:16.311 0.000 UDP 147.32.80.9:50489 -> 147.32.84.131:53 ..... 0 1 300 1
2010-10-22 11:38:16.330 0.000 UDP 147.32.80.9:50264 -> 147.32.84.131:53 ..... 0 1 300 1
2010-10-22 11:38:16.350 0.000 UDP 147.32.80.9:49401 -> 147.32.84.131:53 ..... 0 1 300 1
2010-10-22 11:38:16.370 0.000 UDP 147.32.80.9:58324 -> 147.32.84.131:53 ..... 0 1 179 1
...
```

Figure 2.2: An example of a transfer of 4KB file over SSH by using DNS tunnel

This enables us to create very large variety of actions by combining various captures from database and flow-building scripts. This is the key feature for the insertion of both requests and responses and also side-effects of some types of attacks.

Attacker Goals

The previous section describes actions available to attacker in order to reach goals, which are described here. The PDDL notation is again used to define so called *problems* representing high-level specification of attacker's desired goals. So far, we have defined approximately 10 basic goals.

Problem 5 shows an an example of a problem definition in PDDL language — the propagation of a malware. Each problem consists of a domain specification, an initial state of the world, and a goal to be achieved. Additionally we can specify objects (such as `?SRCIP` or `?DSTIP`) with type specification (e.g. `?SRCIP - SRCIP`) serving as constants in the static plans.

The initial state specifies attacker's prior knowledge about the target network. Using these pre-conditions enable the specification of PDDL problems of attackers either without any knowledge

about the system or with some specified set of information about the target network.

Problem 5 Example of malware propagation problem.

```
(define (problem MALWARE_PROPAGATION)
(:domain INCIDENT_DOMAIN)
(:requirements :typing)
(:objects ?SRCIP - SRCIP ?DSTIP - DSTIP)
(:init (DUMMY_PREDICATE))
(:goal (and (MALWARE_PROPAGATION DSTIP))))
```

Attacker Plans

Final step in realistic attacker's model creation is to generate an execution plan of the attack. As was discussed above, there are two ways to do this. First, we can use static plans representing the most common attack plans or simulating the long-term behavior of known servers. Second, we can use the planner supporting input in PDDL format to generate new plans according to specified domain problem. The planner is a very simple since the shortest possible plan is not needed.

It is important to point out that we had extended plan syntax for the static plans to be able to specify objects of interest. We added parameter specifying if the plan should be repeated without end (`endless=true`). Additionally we extended the typing of the objects allowing more detailed specification. From plan presented in Problem 5 could be seen both previously discussed extensions – for example object `?SrcIP` was specified as IP address from range 0.32.124.0/26. We have also added a delay between actions that can be manually specified or randomized as shown in the example.

Plan 6 Example of a plan (there can be many) solving Problem 5 automatically generated by the planner.

```
(!SSHAccessAttackS endless=false)
(?SrcIP=(IP)0.38.124.0/26)
(?DstIP=(IP)0.32.84.0/26)
(?DstNetwork=(NETWORK)0.32.84.0/24)

(scan_horizontal SrcIP DstNetwork LARGE)
(delay RANDOM(1000,200000))
(scan_vertical SrcIP DstIP SMALL)
(delay RANDOM(1000,200000))
(ssh_bruteforce SrcIP DstIP LARGE)
(delay RANDOM(1000,200000))
(malware_upload SrcIP DstIP)
```

2.2 Events Creation

This section describes a system transforming the unstructured flow labeled by trustfulness values into a set of coherent, well classified events. These events can be later recognized as elements of attacker's plan enabling to guess the attacker's goals and reconfigure the system approximately.

The departure from the prior work is twofold. First, we are no longer interested in mere detection of malicious flows, but in the recognition of attacker's actions, which are later used to infer his plans and goals. Second, the adaptation methods of the Detection layer (outlined in Section 1.2.4) and Section 1.2.5 do not longer rely only on the feedback provided by the challenges, but also and the results of the Game Theoretic layer (Chapter 3), for which we need correct recognition and characterization of opponent's action and plans.

The anomalous flows from the network traffic are grouped into **Events** — sets of flows according to their own characteristics. An event represents symbolic description of a particular network activity (service, behavior) with respect to traffic sources, destinations, and other features. In this section, we describe the process of extracting events from corresponding network traffic and how the classification is assigned to them. The Events are then used as Actions in the model described in Section 2.3. The transcription between Events and Actions is described in Section 2.2.7.

2.2.1 Events Extraction and Clustering

To create an event, we use similarity metrics between individual flows in multi-dimensional feature space. Each flow is identified by five feature values like source and destination IP address, source and destination port and protocol. Beside of these five basic features, each flow contains additional information about the connection, e.g. number of bytes and packets transferred. These all features form the above-mentioned multi-dimensional feature space, and all flows are placed to this space to create clusters of similar entities during multi-staged clustering process.

Flows Pre-Selection

As the first step in the events extraction process, we select the baseline set of flows from whole incoming network traffic that will serve as an input data. Because of the fact that the system concentrates on the malicious behaviors on the network, this input pre-selection skips trustworthy connections or connections with no chance of be part of an event. That means, all flows that are not considered by the system to be legitimate enter the events creation process. The exact pre-selection criteria are described as follows:

Flow φ_i is selected into the input baseline set if and only if:

1. trustfulness of the flow is less than the trustfulness of legitimate threshold, and
2. the total number of flows or bytes (from the whole traffic) with the same source IP address (or destination IP address) as the evaluating flow is greater or equal to threshold describing minimal number of flows or bytes per one event.

First condition selects flows that are considered to be malicious or suspicious by the system, the second one guarantees that flows with no chance of be part of an event are not processed further. The meaning of this condition will be explained later in this section, because it is closely related with the event extraction process itself.

Reducing the total number of processed flows done by the pre-selection also alleviates computational requirements of the event creation process. The detailed information about the event selection process is described in Section 2.2.6.

Creating Elementary Clusters

This first stage of the clustering forms elementary clusters describing elementary network behaviors.

Once a flow passes the flow pre-selection procedure, it is placed into the multi-dimensional feature space as a *flow-cluster* (denoted as ϕ_i^f) together with the rest of already processed flows. The position of a flow-cluster in this space is based on concrete feature values of the processed flow.

The flow-clusters are grouped to *elementary clusters* (denoted as ϕ_j^e – where flow-cluster ϕ_j^f is the former flow-cluster). The distance between a flow-cluster and an elementary cluster is defined by the similarity metrics.

Before we introduce the similarity metrics, we define *bytes similarity condition* which we use in the definition of the similarity metrics itself.

Bytes similarity condition for a flow-cluster ϕ_i^f of the flow φ_i and an elementary-cluster ϕ_j^e is defined as:

$$\mid bytes(\phi_i^f) - avgBytes(\phi_j^e) \mid \leq \min\Big(bytes(\phi_i^f), avgBytes(\phi_j^e)\Big),$$

where $bytes(\phi_i^f)$ denotes number of bytes of the flow-cluster ϕ_i^f (which is in fact number of bytes of flow φ_i), and $avgBytes(\phi_j^e)$ denotes the average value of number of bytes computed from all flow-clusters already in the elementary cluster ϕ_j^e.

As you can see, the bytes similarity condition depends on the size of its input values, the larger are the inputs, the greater difference is accepted. The binary similarity metrics used in this phase of clustering can be described as follows:

The flow-cluster ϕ_i^f of the flow φ_i can be added to elementary-cluster ϕ_j^e if and only if both clusters satisfy bytes similarity condition, have the same protocol and satisfy at least one of the following conditions:

- have the same source IP address and source port
- have the same source IP address and destination port
- have the same destination IP address and source port
- have the same destination IP address and destination port

If there is no similar elementary-cluster found, the flow-cluster ϕ_i^f creates new (its own) elementary-cluster ϕ_i^e.

Event Clusters

The second stage of the clustering process groups elementary clusters into larger cluster entities denoted as event clusters to reveal larger network services or other network behavior. The idea behind this clustering stage is in the fact that the elementary clusters are not always sufficient to describe some services and form only part of them.

An example of such network behavior can be data transfers from one host to another – there is great variability in number of bytes, but all transfers represent the same network behavior. Another example could be web services or peer-to-peer traffic.

Besides these cases, the aggregation of smaller clusters can reveal the true nature of present network behavior. The aggregation process itself is based on a similarity function defined as follows.

Two elementary clusters are similar if and only if they have the same protocol and one of the following conditions is satisfied:

- both clusters have the same average value of number of bytes,
- or both clusters ϕ_i^e and ϕ_j^e have the same source port (or destination port) and general bytes similarity condition is satisfied:

$$\mid avgBytes(\phi_i^e) - avgBytes(\phi_j^e) \mid \leq \ k \cdot \min\left(avgBytes(\phi_i^e), avgBytes(\phi_j^e)\right),$$

  where $k > 1$ is a predefined parameter (in our implementation $k = 100$).

The above-mentioned conditions describe the rules that are used to group similar elementary cluster into event clusters. The event clusters represent second-stage clustering output and can be seen as individual network events. Before the final (third) clustering process, the system first assigns classification to events in order to gain some additional knowledge and thus to increase the event feature space.

2.2.2 Events Classification

This phase of the events creation process provides classification data for a set of events. Classification data consist of:

1. Description of the behavior that the event represents, i.e. "dns tunnel".
2. Severity level of the behavior, on the scale from 0 (least severe) to 10 (most severe).
3. The classification type: generic, specific, or white-list.
4. Measure of the system's confidence in the above.

CNP Auction

The classification of a set of events is provided by running an one-stage auction. The auction is implemented using the Contract Net Protocol (CNP) as follows:

1. The Events Provider agent initiates the auction by sending the set of unclassified events to the Incident Type agents.
2. Each Incident Type agent proposes classification data for each event in the set and sends it back to the Events Provider agent.
3. Upon receiving from all Incident Type agents, or a timeout, the Events Provider agent selects, for each event, the classification data that contain the highest-ranking type value and the highest confidence. This classification data then becomes the resulting classification data for the event.

4. The Events Provider agent sends the results of the auction, including all received proposals, back again to all Incident Type agents.
5. The Incident Type agents recognize the ranking of their proposal and can modify their internal state (e.g. hypothesis values) accordingly.
6. The auction is finished.

Minimum Confidence Threshold

After the auction is finished, the confidence value of the classification is inspected. If this value is lower than a pre-set threshold, an "unknown" classification with severity level 5 is used instead.

Currently this threshold is exactly zero, that is, if at least one Incident Type agent provides a classification proposal with non-zero confidence, then the classification cannot become "unknown". Only if no Incident Type agent can provide a classification, then the event becomes classified as "unknown".

Modifying the threshold sets a trade-off between providing only those classifications where the system is more confident and leaving the rest as "unknown", and between providing all possible hints at classifications, albeit with lesser confidence.

Classification Types

The type of the classification is included in the classification data. The types provide a ranking mechanism, i.e. one further dimension to the confidence value. The generic classification proposals will always be inferior to specific or white-list proposals, and the specific proposals will be inferior to white-list proposals.

1. A generic classification type describes a classification that covers a wide range of behaviors. For example, communication to port 80/TCP can be classified as generic http request; communication with large number of requests to the same machine, but different ports, can be classified as generic port scan-like behavior.
2. A specific classification type is used, where the behavior is defined precisely. For example, generic port scan-like behavior with TCP protocol and exactly 40 bytes per flow can be classified as a specific port scan.
3. A white-list classification type is used when the CAMNEP operator explicitly wishes to not report certain types of behavior. As there are no Incident Type agents providing this classification type by default, they must be configured manually.

2.2.3 Incident Type agents

An Incident Type agent is responsible for providing a proposal on what the event could be classified as. For each processed data-set, the Incident Type agent is provided with the full set of events (i.e. sets of clustered flows), and also the results of each auction evaluation (i.e. what other agents said about the events, who won the auction). The Incident Type agent is free to use and/or store all this information to provide the best classification for the events.

Currently, two types of Incident Type agents are employed: the stateless agents and the stateful service-monitoring agents.

Stateless Incident Type agents

These agents provide their classification proposal by using a filter-matching mechanism on each event. The classification data is pre-configured, with the exception of the confidence value, which reflects the match quality of the configured filter.

The filter is a sequence of rules providing constrains for the behavior represented by the agent. Using the examples from above, a very simple rule could be a constraint on the communication to TCP port 80. For this rule, two matching strategies can be used:

1. Boolean: the rule matches if and only if all flows have destination port 80 and TCP protocol.
2. Fuzzy: the rule match value is proportional to the number of flows in the event that contain these features.

Currently, both strategies are supported and useful for various types of behavior.

The rule specification can include wide range of constrains:

1. Flow features (addresses, ports, protocols, sizes).
2. Number of flows in the event.
3. Statistics over the features (number of distinct values, entropies, averages).

When multiple (n) rules are used in the filter, the match quality of the filter is a product of individual rule match qualities: $Q^{(\text{Filter})} = \prod_{i=1}^{n} Q_i^{(\text{Rule})}$.

2.2.4 Stateful Service-Monitoring Incident Type agents

The Stateful Incident Type agents (SITA) are more complex than the stateless version. SITAs are also defined by the static filter specifications as stateless agents, but unlike them, stateful agents maintain their own models of network traffic services (each agent models one service) giving them awareness of the current network state from the long term perspective. In this section, we describe the details of SITA modeling process and show our approach of modeling individual network services.

As stated above, each SITA is defined by static filter specification described in Section 2.2.3 describing the type of service, its properties, purpose, and characteristics. It is important to point out that each SITA models only one particular type of network service. From this reason it uses information from events that match only this predefined filter specification and other events are not considered.

From each event (that positively matches the filter) the SITA extracts source(s) of this event (which is the combination of source IP address and source port) and feature values, such as set of destination IP addresses, number of flows, etc. SITA maintains a separate model for each source that has been extracted from accepted events. This model consists of two types of information:

1. modeled feature space
2. hypothesis value

Modeled Feature Space

To model the event sources, we use several features like:

- set of destination IP addresses to model the address space of each source,
- number of flows or bytes to model the traffic volumes.

For example, model fit in address space is based on distinguishing between three degrees of communication:

- Private - modeled source communicates with single host
- LAN - modeled source communicates within single A-class subnet with limited number of targets
- Internet - modeled source has no communication restrictions

By maintaining a set of targeted hosts, SITA is able to detect any changes in degree of communication of each modeled event source.

SITA uses these feature values to compute a similarity between an event and existing model of feature space of the corresponding source(s) presented in this event. We will call this similarity between an event and corresponding SITA model as *model fit* $\in [0, 1]$ (unlike *event match*, which is the similarity between an event and SITA static filter).

The model is used to determine the difference between behavior tendencies of the source through the time and the actual event, which exhibits some certain degree of anomaly (or degree of expectedness).

Hypothesis Value

Beside modeled feature space, each individual model contains also *hypothesis value* $\in [0, 1]$, which describes the belief that corresponding source really uses SITA's service or behavior (e.g. if SITA represents a web server, the hypothesis values quantify its belief that modeled sources are truly web servers). We use sequential hypothesis testing mechanism for update (where each incoming event modifies the hypothesis values of its sources), and there is also slight decrease in time when no event of that source occurs.

When SITA inserts new source into its model, the initial hypothesis value is set to 0.5. As system receives new events, this value changes according to the simple principles.

Hypothesis value of a source from an event:

- is increased if and only if SITA wins this event in the auction

- is decreased if and only if SITA send a bid and lost in the auction
- is slightly decreased in time (per dataset)

In case of more increases (or decreases) per one dataset, the first change is applied by multiplying current hypothesis value with rate $c_1 \in [1, 2]$ (or $c_1 \in [0, 1]$), but the second change is applied by multiplying already changed value with rate of smaller impact $c_2 \in [1, c_1)$ (or $c_2 \in [c_1, 1]$).

State diagram of SITA modeling process is illustrated on Figure 2.3.

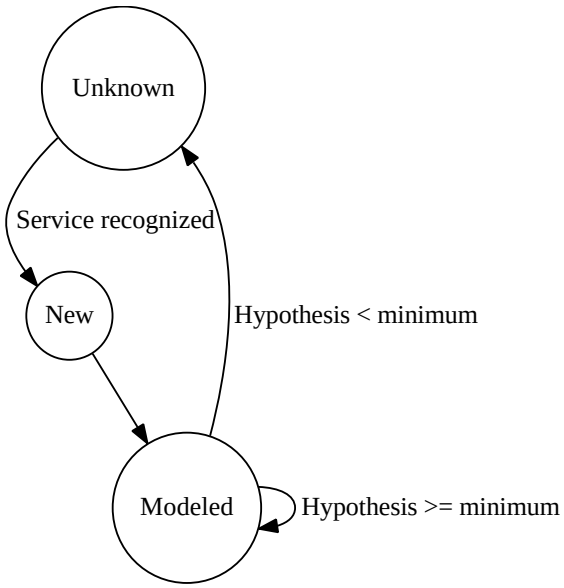


Figure 2.3: State diagram of Stateful Incident Type agent modeling process.

Classification bid

In this Section, we summarize the whole process of event classification by using Stateful Incident Type agents. The event classification bid for the auction is computed from three different values:

1. quality of match with predefined static filter specification
2. size of model fit with corresponding inner model of feature space
3. size of hypothesis value

The whole SITA classification process can be described in the following steps:

1. SITA receives set of events and performs static filter specification match, where each event is compared with predefined filter describing SITA's specification. Only events with positive match are processed further.

2. From each event, the algorithm extracts event sources and computes model fit to determine the deviation from modeled long term behavior of the sources. In case of more sources presented in the event, the algorithm selects the minimal model fit to emphasize unexpected behavior.
3. The algorithm combines model fit with current hypothesis value to determine final bid offer. In our implementation we use 60% for the model fit and 40% for the hypothesis value. The final bid is sent back to Events Provider.
4. Once the auction is complete, the results are sent back to SITA to perform an update of both hypothesis values and modeled feature space.

2.2.5 Classification Clustering

This phase is the last in the events creation process. It is responsible for clustering events using all available information, specifically the event classification.

An important concept that can be derived from the classification is the *originator* of the behavior.

If the event is classified as request, then the originator is the set of source addresses in the event. If the event is classified as a response to a previous request, then the originator is the set of the destination addresses in the event. The specifics of what is a request and what is a response are described in the earlier sections. Note though, that a response to a request event does not necessarily require that the request event has been seen before.

The clustering based on classifications considers any number of events to be part of the same behavior if and only if they have the same classification and originator. Such events are joined together into a set of flows that is then presented as one larger event. If at least one of these joined events was previously considered as malicious (i.e. its flows were in the malicious zone of trust), then the larger event is also considered as malicious.

2.2.6 Events selection process

The events selection process is responsible for filtering a set of events with regard to a quality/quantity trade-off. The resulting set is used as the system output for automatic reporting, GUI display, etc.

Given a set of flows observed in traffic, the trust value for each flow is determined in the detection layer, and the set is then divided into three zones – malicious, neutral, and legitimate. The events creation process will take into consideration all flows from the malicious and neutral zones. The full output from these zones typically contains hundreds of events, which is too many to report for each data-set (typically 5 minutes worth of traffic).

Parameters

There are many parameters affecting the trade-off between the quality, the quantity, and the minimum event size. These parameters can be used to fine-tune the output of the algorithm, namely:

1. Which events are considered important?

2. How important is the fullness of the automated classification with regard to signal to noise ratio?

3. What amount of traffic is considered negligible?

Events quality For each event, we can establish an estimated information value that the event in question provides to the human operator. This information value, averaged over a set of events, represents the quality of the set: $Q = \sum_{i=1}^{n} V_i/n$.

Currently the value V_i is defined as binary value, i.e. the event is important or not.

The malicious zone is likely to contain many true negatives (malicious flows classified as malicious). Not including events from this zone imposes the risk of generating false negatives. The events in this zone are implicitly regarded as important, i.e. information value for events in this zone is $V_i = 1$.

The neutral zone is likely to contain many false positives (legitimate flows classified as malicious), i.e. noise, that provides little information value. Including all events that have been created by the events creator process is likely to yield a set with poor quality. To avert this, we define an event in this zone as important only if it satisfies one or both of the following conditions:

1. It represents a behavior that is classified as severe, i.e. severity > 5 on the scale from 0 to 10.

2. It represents a significant portion of the traffic, i.e. the minimum size thresholds (see below) are exceeded by a factor of 10.

In our experiments these conditions serve well to penalize false positives enough to be excluded from the selected set of events.

The legitimate zone is unlikely to contain true positives. Classification of legitimate traffic is only of minor concern for the system, which is why the legitimate zone is exempted from the events creation process altogether.

Events quantity As mentioned in the previous paragraph, including too many events will result in poor quality. On the other hand, including too few events would result in large amounts of non-categorized traffic, reducing the usefulness of the events system as whole.

In our experiments, we sought the amount of traffic that should be categorized, which, when exceeded, results in rapidly diminishing quality of the resulting events set.

The value has been determined to be 98% of the number of malicious flows in the traffic and 90% of the number of bytes summed from all malicious flows included in the traffic. In other words, we can afford not to classify at most 2% of the malicious traffic, flows-wise and at most 10% byte-wise, without significant degradation of the system usefulness.

These ratios can be adjusted arbitrarily, resulting in more or less emphasis on fullness of the automatic classification of the traffic. We'll call this the "at most X% of malicious traffic not classified rule".

Minimum size of an event The "at most X% of malicious traffic not classified rule" from the events quantity section, if followed rigorously, could theoretically result in selection of a very large number of events that contain very few flows or very few bytes. These events typically represent either stray flows that haven't been linked to any other event (i.e. behavior on the network) by the events creation process, or they represent an attempt to overload the system with noise.

We regard these as waste – inclusion of large numbers of such small events would provide very little information to the operator. They would severely reduce the quality of the selected set of events. We apply a thresholding technique, i.e. we seek to include only events that meet a minimum size requirement.

Again, the thresholds can be set to arbitrary values, resulting in various levels of compromise between full classification and the signal to noise ratio of the classification.

Through experimentation we've arrived at the minimum volume threshold to be statically 20 kilobytes in size, which is enough to catch all but the smallest stray data transfers. The minimum flows threshold is dynamic with regard to the number of all flows processed (i.e. in the whole traffic). The formula is $\max(10, F/2000)$ flows, where F is the number of all flows processed.

For an event to be considered for inclusion, any of the thresholds can be exceeded – either the behavior has transferred a non-negligible amount of information in a single connection (e.g. data leaks, tunnels, brute forcing, etc.), or it represents a non-negligible amount of distinct connection attempts (e.g. scans, service failures, and again brute forcing, etc.).

The thresholding takes precedence over the "at most X% of malicious traffic not classified rule", and in some cases, can result in violation thereof. In these cases, the system was unable to find a behavioral connection between the flows. It will leave the flows unclassified and will output them as a single block, which can then be analyzed manually using the flows analyzer.

The algorithm

The events selection algorithm runs in several phases.

The first phase occurs at the very beginning of the events creation process. If the events creation process determines, through statistics on the processed traffic, that a certain flow cannot be clustered with another flows to form a basis of a behavior, and it does not meet the minimum size requirement, then the flow is exempted from events creation altogether. This is an optimization of the process where such a flow would be considered as an event, classified, and then that event dropped on the basis of not meeting the minimum size requirement.

The second phase occurs after the events creation process has finished, providing a set of classified events. Primarily, the algorithm will include all events that are considered important and meet the minimum size requirements.

If at this stage the "at most X% of malicious traffic not classified rule" is not already satisfied, a greedy algorithm is applied to include as few unimportant events as possible to meet the rule. These also must satisfy the minimum size requirements.

The rationale for this algorithm comes from the fact that the information value of an event is currently binary, i.e. it provides a piece of important information or not:

1. We first include all important events to maximize the total information value.

2. We then include unimportant events on a largest-first basis, to degrade the total information value as little as possible.

This algorithm is sufficiently fast, as it requires at most $O(N)$ operations, where N is the number of events.

2.2.7 Action recognition

As we have mentioned in the introduction of this section, the events generated using the events extraction and labeling process are used in game-theoretical model as a sequence of attacker's actions. However, the labeled events cannot be used as actions directly because of the incompatibility between event's types and action's labeling. This incompatibility reflects the different needs of the human operators of the system and the automated game-playing methods. Therefore, we have to introduce mapping between labels assigned by the labeling algorithm and action's names defined in the attacker's model discussed in Section 2.3.

The mapping between event's labels and action's names can be characterized as $M : N$ relation. There is an event which maps to more than one action and on the other hand more events map to single action. For example, event labels `http request`, `https request`, `https request`, and `https response` map to single action — `WEB_ATTACKS` and events classified as `data transfer (tcp)` can represent one of actions `CONNECT_TO_HOST`, `CONNECT_TO_C2` and `UPLOAD_DATA_TO_C2`. Next, we have to select only the subset of labels which refer to actions defined in the attacker's model used in context of the current work. This means that a subset of event labels (as produced by the classification/labeling process) does not map to actions in the game theoretic model. For example, events classified as `new unexpected service`, `new unknown client`, etc. labeled by stateful ITA agents do not have any matching actions. Thus, at this stage, we have decided to use only the events labeled by stateless agents. These agents produce classified events such as `DNS request/response`, `vertical TCP port scan`, etc. which directly map to the actions in the model. The result of stateful event classification is used as a negative filter, i.e. it reduces the number of events to consider, and therefore reduces the computational complexity of the problem. The complete visualization of the mapping is shown in Table 2.1.

2.2.8 Experiments

We have made several experiments to measure the accuracy of the events creation and classification process. Normally the input for the events creation and classification process does not contain any simulated flows, but solely for the purpose of this experiment we have modified the system to include them. This way we were able to evaluate hypothesis evolution on a challenges simulated by the plan executor agent. This means that we measured how these two layers perform against each other.

Figure 2.4 shows that hypothesis value of a SITA representing http server grows rapidly from the uncertain value of 0.5 to the value near 1, representing near certainty. The evaluation is performed on the simulated flows that were generated by Plan executor from the repetitive Plan 7 containing only one web server-like behavior action.

Figure 2.5, shows evolution of hypothesis of SSH server SITA during several large concurrent SSH brute-force attacks against one IP address. There were few attackers that were simulated by plan executors. Each of them was executing a static plan starting with a large scan of the network

data transfer (tcp)	{CONNECT_TO_HOST, CONNECT_TO_C2, UPLOAD_DATA_TO_C2}
data transfer (udp)	{CONNECT_TO_HOST, CONNECT_TO_C2, UPLOAD_DATA_TO_C2}
dns request	{DNS_REQUESTS}
dns response	{DNS_REQUESTS}
dns tunnel client request	{DNS_REQUESTS}
dns tunnel client response	{DNS_REQUESTS}
ftp-data request	{UPLOAD_DATA_TO_C2}
ftp-data response	{UPLOAD_DATA_TO_C2}
heavy dns use	{DDOS_TO_SPECIFIC_SERVICE, DDOS_TO_HOST}
http request	{WEB_ATTACKS}
http response	{WEB_ATTACKS}
https request	{WEB_ATTACKS}
https response	{WEB_ATTACKS}
icmp traffic	{HORIZONTAL_PING_SCAN}
periodical polling requests	{CONNECT_TO_C2, UPLOAD_DATA_TO_C2}
port scan (horizontal, tcp)	{HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE, HORIZONTAL_PING_SCAN}
port scan (horizontal, udp)	{HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE, HORIZONTAL_PING_SCAN}
scan-like behavior (horizontal, tcp)	{HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE, HORIZONTAL_PING_SCAN}
scan-like behavior (horizontal, udp)	{HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE, HORIZONTAL_PING_SCAN}
scan-responses-like behavior (horizontal, tcp)	{HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE, HORIZONTAL_PING_SCAN}
scan-responses-like behavior (horizontal, udp)	{HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE, HORIZONTAL_PING_SCAN}
port scan (vertical, tcp)	{FINGERPRINTING, PORT_SCAN}
port scan (vertical, udp)	{FINGERPRINTING, PORT_SCAN}
scan-like behavior (vertical, tcp)	{FINGERPRINTING, PORT_SCAN}
scan-like behavior (vertical, udp)	{FINGERPRINTING, PORT_SCAN}
scan-responses-like behavior (vertical, tcp)	{FINGERPRINTING, PORT_SCAN}
scan-responses-like behavior (vertical, udp)	{FINGERPRINTING, PORT_SCAN}
proxy request	{WEB_ATTACKS}
proxy response	{WEB_ATTACKS}
p2p-like behavior (tcp)	{CONNECT_TO_C2, UPLOAD_DATA_TO_C2}
p2p-like behavior (udp)	{CONNECT_TO_C2, UPLOAD_DATA_TO_C2}
scan-like behavior	{HORIZONTAL_PING_SCAN, HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE}
scan-responses-like behavior	{HORIZONTAL_PING_SCAN, HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE}
smtp request	{SEND_SPAM}
smtp response	{SEND_SPAM}
smtps request	{SEND_SPAM}
smtps response	{SEND_SPAM}
ssh cracking	{BRUTEFORCE}
ssh request	{CONNECT_TO_HOST}
ssh response	{CONNECT_TO_HOS}

Table 2.1: Events – actions $M : N$ mapping.

Plan 7 Static web server plan example.

```
(!webServer endless=true)
(?ServerIP=(IP)0.32.80.120)
(?ClientsSubnet=(IP)0.32.84.0/26)

(webServer ServerIP ClientsSubnet LARGE)
```

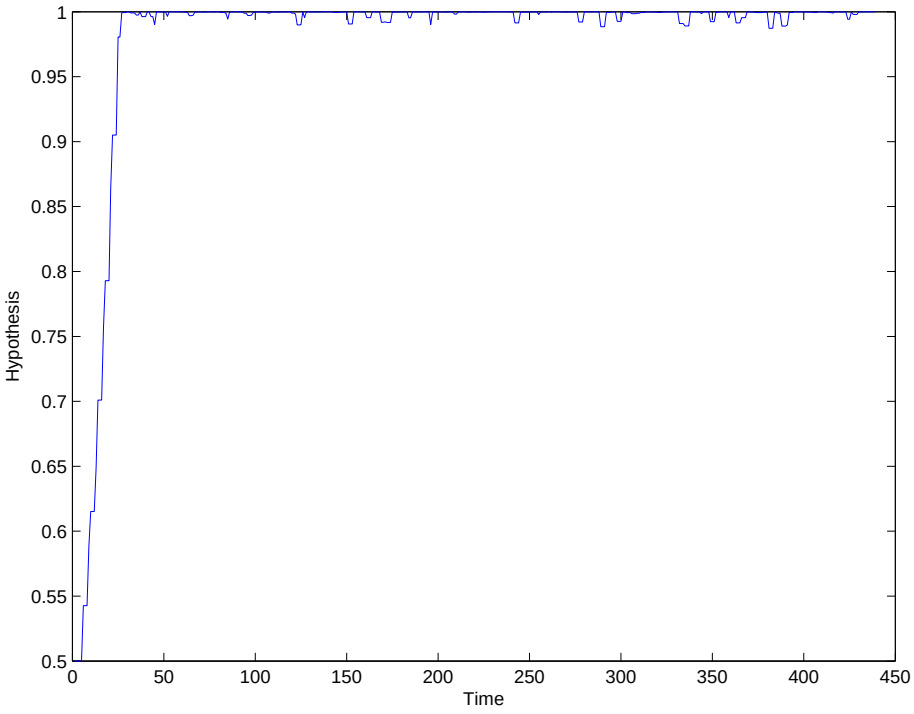


Figure 2.4: Hypothesis growth of the http server stateful ITA on a flows that were generated by the Plan executor that simulates web server.

and then trying SSH brute-force attack on predefined IPs. The graph shows the hypothesis value of the SSH sever ITA that corresponding to the attacked IP. It can be seen that the hypothesis decreases from its initial value of 0.5, albeit not monotonically. This is caused by the competition of the stateful ITA representing SSH server and stateless ITA corresponds to SSH cracking, which eventually wins.

2.3 Attacker's Model in the System

In order to recognize attacker's plans using the game-theoretical approach we have to be able to model his behavior. In Section 2.2 we have presented an action extraction process which allows us to detect single action of an attacker. However, to recognize attacker's plans we have to know much more than individual actions. Thus, we have to be able to predict possible actions in next step.

To solve this problem we present technique for plan representation based on a PDDL domain. We assume that attacker is able to perform only these actions that are defined in the PDDL domain. Moreover, the sequence of attacker's actions has to fulfill the constrains defined by this domain. Therefore, using this domain we are able to model attacker's behavior. Note that the domain described in this section can be used not only for plan recognition but for advance attacker's simulation as well (See Section 2.1).

In this section we will at first discuss attack graphs as the basis for the PDDL domain definition. Next, we will present the domain created using these graphs.

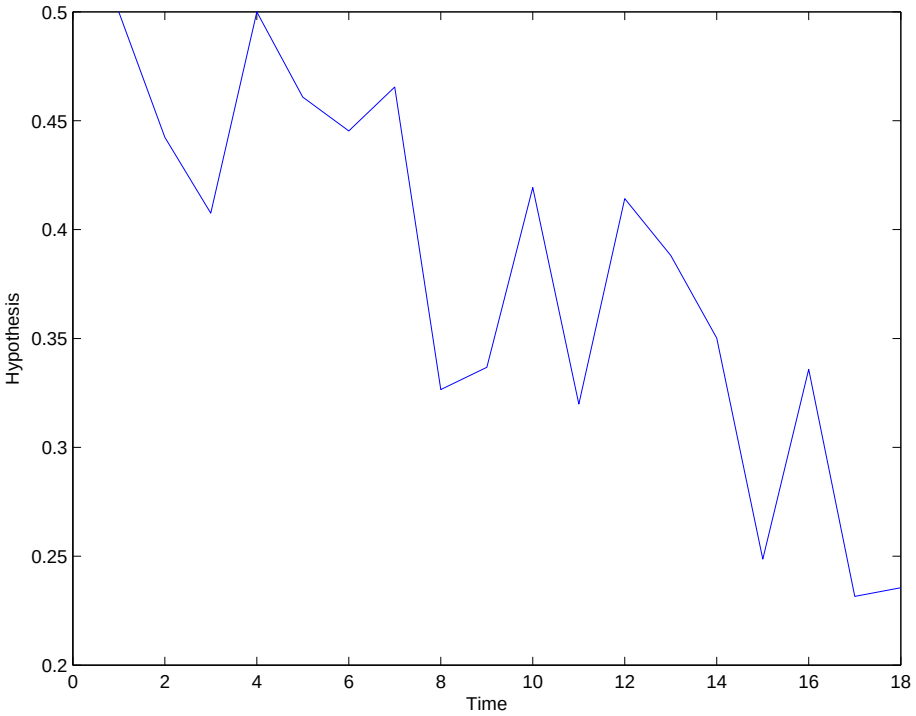


Figure 2.5: Evolution of hypothesis of a stateful ITA when inserting flows that represent SSH access attack plan.

2.3.1 Attacker graphs

Based on behavior of the real network attackers observed during last decade we are able to identify similarities and common steps which most of the attackers have to perform to be able to fulfill their goals. From these observations, we are able to create a graph representing different stages of most of the common attacks. In this section we will introduce the proposed attack graph structure and describe specific stages of the attack graph in more detail.

From our point of view, we define 5 different stages: information gathering (reconnaissance), attack, maintaining access, spread and call home. Relations between these stages can be seen on Figure 2.6. As this figure shows, the attacker can return to reconnaissance stage after any part of the attack in case that he found some interesting information about targeted network which enables him to modify goals of his attack or start a whole new attack. This fact is represented by arrow returning from all stages back to the reconnaissance stage.

Note that in all diagrams the parallelogram-shaped nodes denote the whole stages of an attack. Next, the oval-shaped nodes denote internal action which is not visible to the defender through the prism of the NetFlow data — e.g. choosing a target or executing commands on targeted host, etc. The rectangular nodes appearing in all of the diagrams correspond to the part of an attack which is observable as a network action and can be therefore traced to one or more flows in the NetFlow data. Last, the octagon-shaped leafs represent the subgoals of a stage.

At first, the attacker has to perform the *information gathering* displayed on Figure 2.7. During this stage he is trying to discover information, such as information about the company, their defenses, IP addresses of important servers, etc., the attacker is choosing target. To obtain all this necessary information to penetrate the defense of the targeted network he can use various techniques, such

as *horizontal ping scanning*, *horizontal scans for specific service*, *web crawling*, etc. He can also use some techniques of social engineering such as *phishing* to obtain access to important accounts on servers.

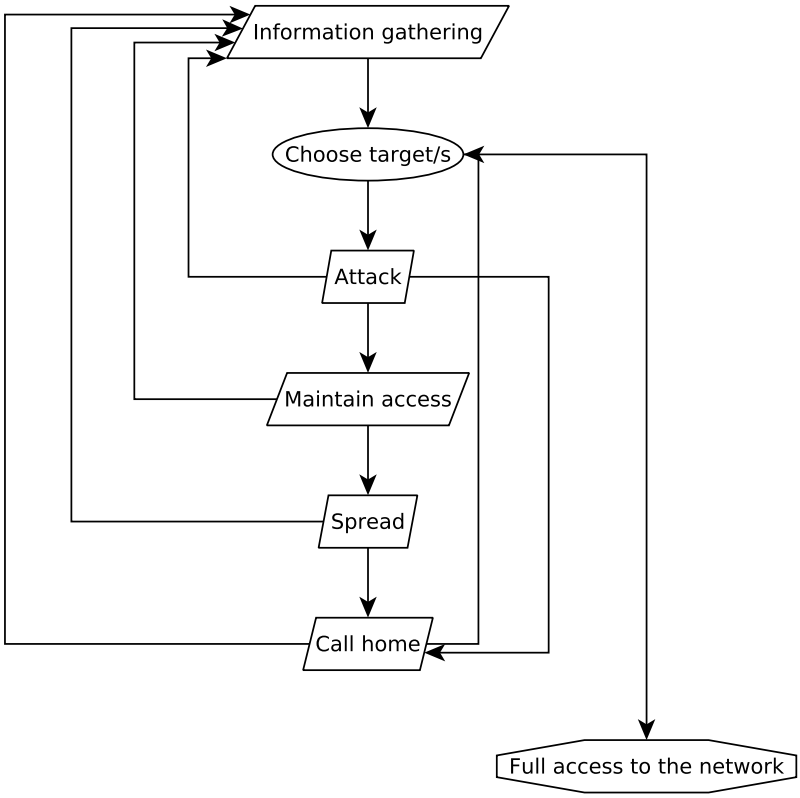


Figure 2.6: Global graph representing the attack as a whole.

Next, when he possesses all necessary information he is ready to choose his target and then perform the attack. In the attack stage the main attacks such as bruteforce or DDoS are performed to gain access to targeted machines or disable specific service. In this stage, the attacker is most visible to the IDS system. As it needs to perform an attack from outside of the network, he has to generate some amount of NetFlows that will be processed by the CAMNEP system. This stage is shown on Figure 2.8.

If the exploit of the targeted system was successfull, the attacker proceeds further. He tries to *maintain* persistent access to the system by means of *backdoors*, *sysproxy*, periodic reverse calls or use of covert channels, often concealed from the OS by the use of *rootkit*thus gain the ability to establish a new connection to the system at will.

When the malware is installed into targeted network, the attacker is able to *spread* the malware further into the other hosts within the targeted network to affect more computers and search for more interesting data.

In the last stage, the malware is already spread over the target network and awaits a trigger or a signal from C&C[1] structure. When it receives this notification through one or more possible channels, it starts its activity and sends collected data to C&C server and therefore achieves its objective. We assume that the data sensitive data is harvested by means of key loggers, targeted

[1]Command and control

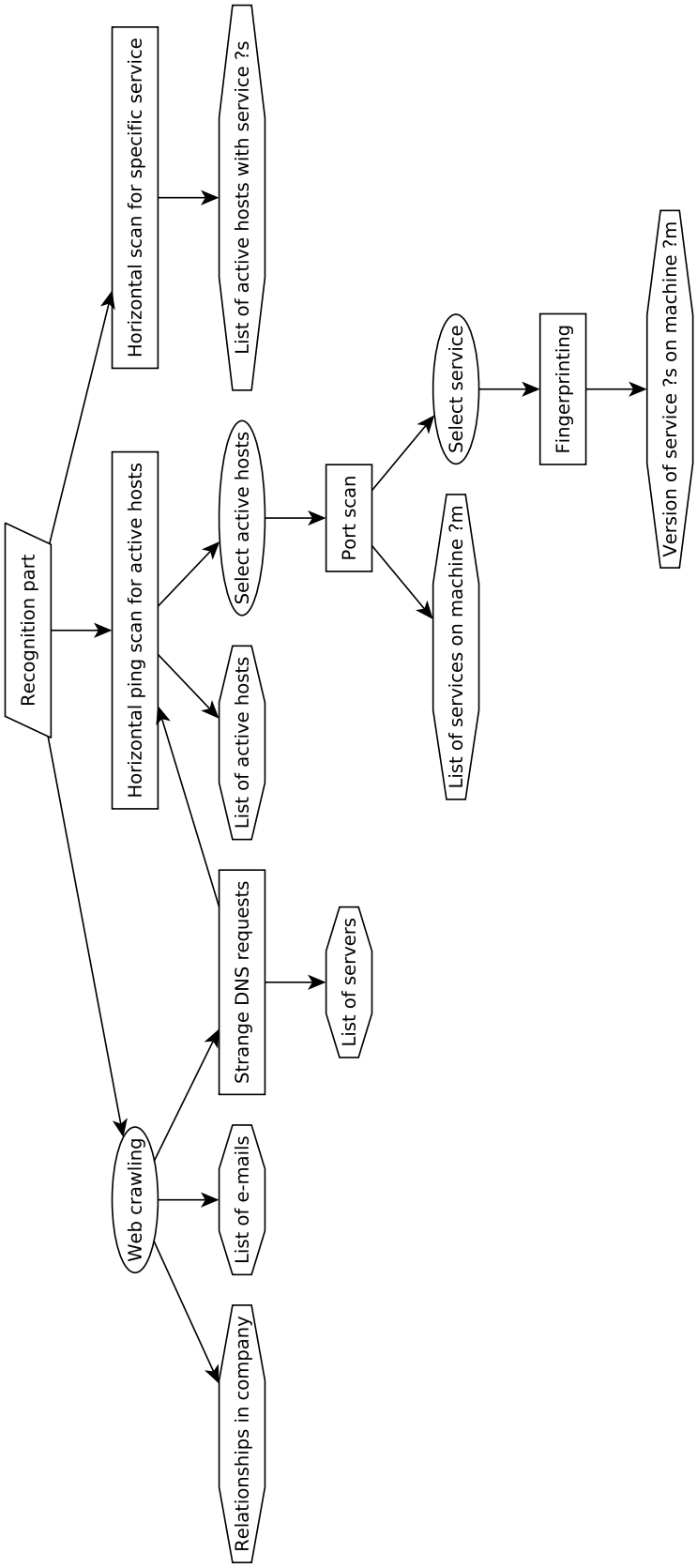


Figure 2.7: Detailed description of the recognition stage of an attack.

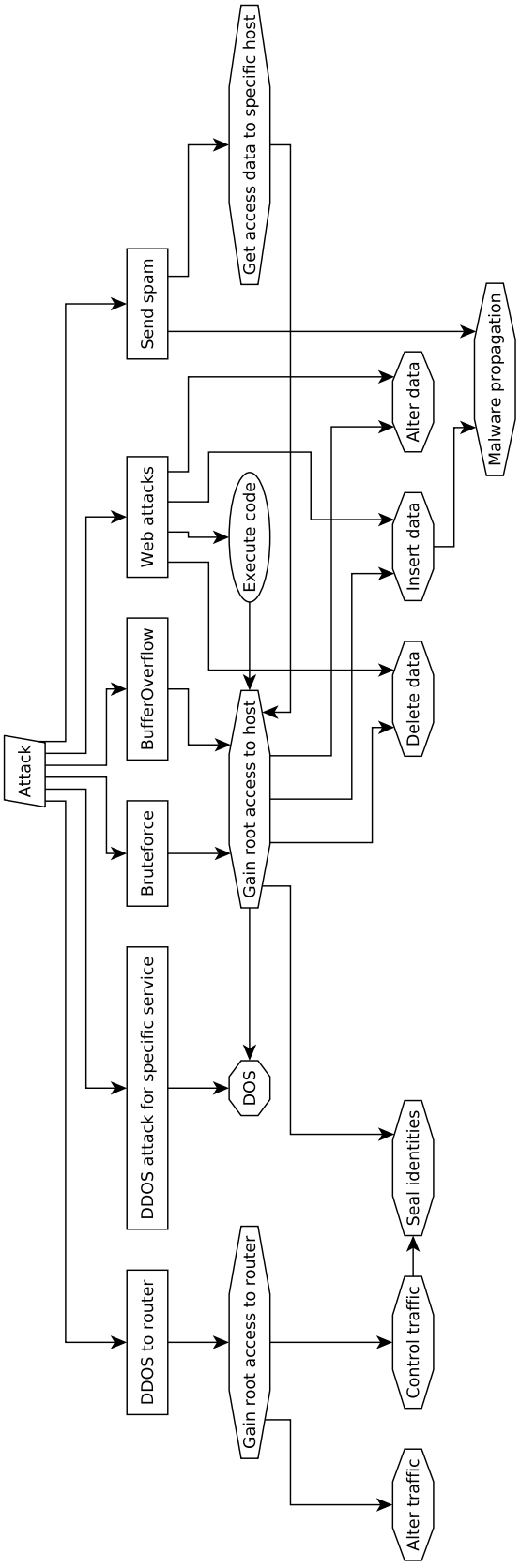


Figure 2.8: Main part of the whole attack. Possible techniques to perform attack and gain access to the targeted network and possible goals.

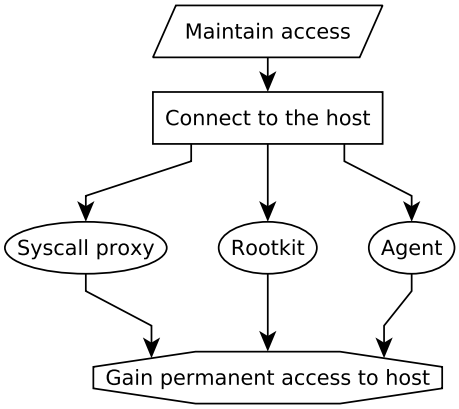


Figure 2.9: Maintaining access to affected hosts in the network.

copies of used/modified files, local keyword-based search for specific documents or by copying the documents from default/recently accessed locations.

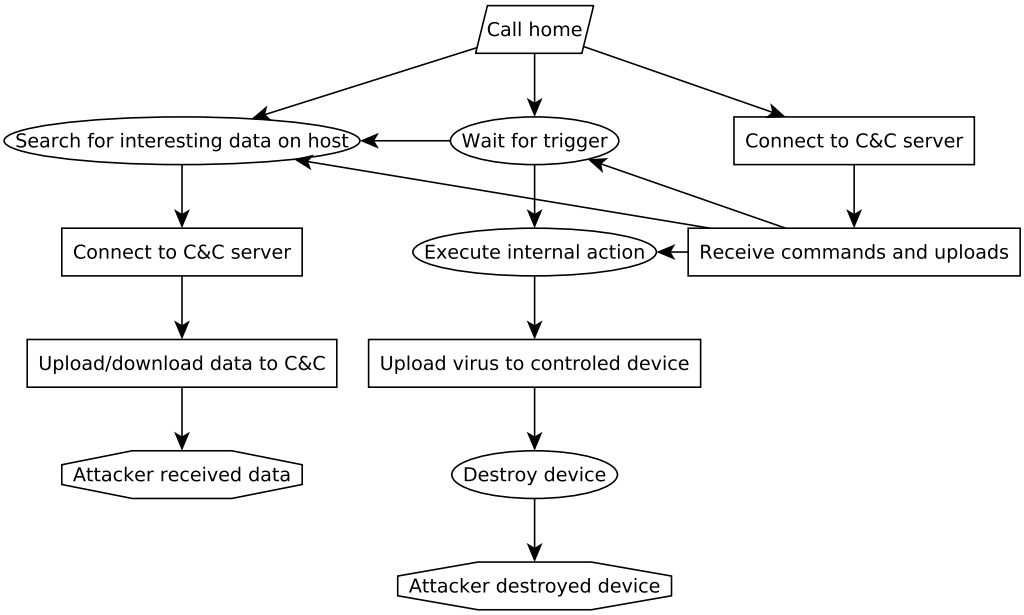


Figure 2.10: Call-home stage. After the malware is successfully spread into network it is waiting for trigger or searching for interesting data which will be sent to the attacker.

2.3.2 Definition of the PDDL Domain

In previous section we have introduced attack graph as a formalism used to describe usual behavior of the attackers. Using these graphs, we have defined plans suitable for common attacks performed against protected networks. We can use this knowledge to define a domain which allows us to model much wider attacker's behavior and therefore automatically generate far richer family of plans. However, these plans blindly generated by traversing this domain do not have to be necessarily reasonable (i.e. optimal) and therefore we have to add some level of knowledge into the algorithm

to generate more realistic plans of attacks. To solve this problem we have used game-theoretical approach which is described in Section 3.3.

As we have stated above, we have used the attack graphs as the basis for definition the planning domain. We have defined the attacker's actions which are represented by nodes in diagrams 2.6, 2.7, 2.8, 2.9 and 2.10. We have decided to limit the number of actions because of the computational complexity. As we are not able to see any actions that don't result in network activity (and corresponding NetFlow data), all the actions denoted by oval-shaped nodes are not modeled. These two limitations restrict the model to actions which are visible as network connections.

In order to define this domain, we have used the PDDL language briefly described in Section 2.1.2. Each action can have *preconditions* which have to be fulfilled before this action can be applied and has *effects* which affect the model after the action is applied. Preconditions and effects are defined as a set of *predicates*. Before any action is performed, the predicates defined in its preconditions need to be active. Once the action is performed, the set of predicates defined in effect becomes active. Using this approach, we can easily model changes of the state of the world defined by this domain.

In examples 8, 9, 10, 11 and 12, we illustrate a plan with the goal of infecting a hardware device (such as robotic arm, electricity generator or similar device) controlled by a specific host in the network. The success would enable the attacker to gain control of the device or even render this device inoperable. The first action *HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE* has no preconditions and therefore attacker modeled by this domain can perform this action. After performing this action predicates *LIST_OF_ACTIVE_HOSTS_KNOWLEDGE* and *SERVICE_ON_ACTIVE_HOST_KNOWLEDGE* become active. The predicate *SERVICE_ON_ACTIVE_HOST_KNOWLEDGE* is listed as precondition in next action — the *BRUTEFORCE*. When the *BRUTEFORCE* action is executed predicates *ACCESS_TO_SERVER_KNOWLEDGE* and *CAN_STOP* become active. Note that predicate *CAN_STOP* indicates that this state of the world could be taken as final and no other action has to be performed. However in our example we have three more actions which attacker carries out to fulfill his goals. Note that, the defined PDDL domain is able to model much richer plans than the presented one.

Action 8 Example of horizontal scan for specific service – e.g. SSH service

```
(:action HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE
:parameters()
:precondition ()
:effect (and
(LIST_OF_ACTIVE_HOSTS_KNOWLEDGE)
(SERVICE_ON_ACTIVE_HOST_KNOWLEDGE))
)
```

Action 9 Example of bruteforce attack for specific service

```
(:action BRUTEFORCE
:parameters ()
:precondition (SERVICE_ON_ACTIVE_HOST_KNOWLEDGE)
:effect (and (ACCESS_TO_SERVER_KNOWLEDGE) (CAN_STOP))
)
```

In order to build the global game-theoretical model, the price/value of each plan needs to be evaluated in order to assess the importance of the plan. The value V is defined as sum of cost of

Action 10 Example of establishing access to targeted host in the network

(:action **CONNECT_TO_HOST**
:parameters ()
:precondition (ACCESS_TO_SERVER_KNOWLEDGE)
:effect (and (PERMANENT_CONNECTION_KNOWLEDGE) (CAN_STOP))
)

Action 11 In this action malware installed into targeted host in the network establishes connection to the C&C server to obtain new commands or updates.

(:action **CONNECT_TO_C2**
:parameters ()
:precondition (ACCESS_TO_SERVER_KNOWLEDGE)
:effect (and (UPDATES_RECEIVED) (COMMANDS_RECEIVED))
)

all active predicates P_A.

$$V = \sum_{p \in P_A} C_p \tag{2.1}$$

where $C_p \in \mathcal{R}^+$ is a cost of the predicate p that is set according the current network security policy. Thus, the value V of each of the plans describes the overall security risk.

To generate plans from defined PDDL domain we have to traverse through the domain. The traverse algorithm starts with a default world state, where there are no active predicates and explore the domain in the depth-first search manner. In the first step, only actions that have no preconditions are applicable. Next, it takes all actions applicable to the current state of the world, i.e. it selects an action with active predicates. Next, it prunes actions which gives no more information and therefore are useless for attacker. This way we are able to generate large set of various plans. That are used in game-theoretical model for attacker's behavior estimation as described in more detail in Section 3.3

2.4 Confusion Matrix Computation

In order to model the detection capability of the system on the level that can be represented and used in the game-theoretical reasoning, we will build the confusion matrix which quantifies the joint error function of the anomaly detection and event classification components of the CAMNEP system.

Action 12 Example of propagating virus downloaded from C&C server and propagating this virus into device controlled or managed by targeted host.

(:action **UPLOAD_VIRUS_TO_CONTROLLED_DEVICE**
:parameters ()
:precondition (COMMANDS_RECEIVED)
:effect (and
(VIRUS_UPLOADED_TO_CONTROLLED_DEVICE)
(GAIN_CONTROL_TO_DEVICE))
)

In this section we will detail the technique for the estimation of detection probabilities that are used in game theoretical model (See Section 3.3 for more details) to select the optimal defender's action. The probability P defined in Formula 2.2 represents the probability that system observes action o_k (so called *observation*) when attacker performed action a_j and defender used action d_i. Note, that observations o_k and attacker's actions a_j are from the same set of actions defined by PDDL domain described in Section 2.3.2.

$$P\left(o_k|d_i, a_j\right). \tag{2.2}$$

2.4.1 Estimation of the Confusion Matrices

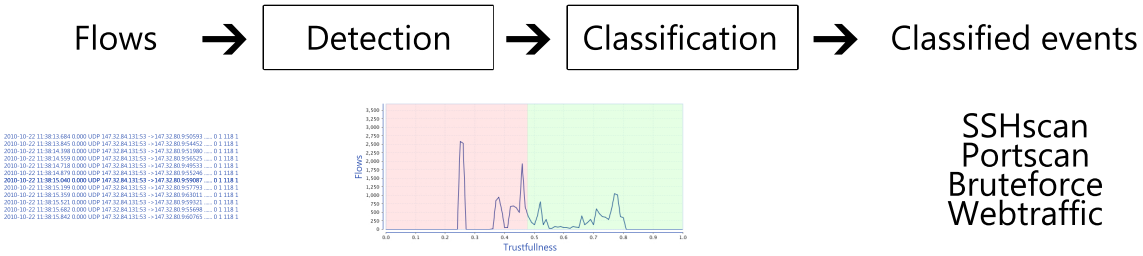


Figure 2.11: Scheme of the detection process.

The detection process is performed in two stages (See Figure 2.11) that affect the detection and correct classification of attacker's action a_j. The first stage, on Figure 2.11 denoted as *Detection*, performs anomaly detection (described in Section 1.2.4). There is a set of anomaly detection methods evaluating each attacker's action. Next, results from all methods are aggregated into single final value. Different methods of aggregation represent defender's actions in our definition of the game. Note, that in this part of detection process actions are represented as sets of flows without label. The detection stage evaluates for each of those flows trustfullness value. The trusted flows are considered legitimate and untrusted flows as malicious. Additionally, the first layer compute the position of the threshold between legitimate and malicious traffic and only the malicious traffic is passed to action extraction and labeling phase (See Section 2.2).

We can estimate probability of detection as follows

$$P\left(a_j|d_i\right) = \frac{|a_j^{\text{mal}}|}{|a_j|}. \tag{2.3}$$

where $|a_j^{\text{mal}}|$ is number of flows generated by action a_j which were labeled as malicious when aggregation function (i.e. defender action) d_i was used and $|a_j|$ is number of all flows from action a_j. The probability $P(a_j|d_i)$ represents the probability that action a_j is denoted as malicious and passed to the second phase when aggregation function (i.e. defender action) d_i was used. Note that, the estimation of the probability $P(a_j|d_i)$ is computed using preclassified sets of flows called challenges (see Section 1.2.4).

The second process, the actions extraction and labeling stage — on Figure 2.11 denoted as *Classification* — receives only the flows labeled by detection stage as malicious and produces classified actions which are used as observations of attacker's actions. The performance of the classification process affects the output of the whole system and therefore has to be included in observation probability $P(o_k|a_j, d_i)$ estimate.

The probability of correct classification is estimated as $P(o_k|a_j)$. To obtain this estimate, we have passed all captured flows to classification phase and processed them. Then, we have computed the classification error, which represents the classification probability. This experiment was performed on a 14-day NetFlow capture from the Czech Technical University network that was manually classified in advance. The classification was made by a network analysis expert and flows were labeled with matching actions from the PDDL domain.

Using result from this experiment we were able to create a confusion matrix shown in Table 2.2.

0.6817	0.0023	0.2912	0.0	0.0	0.0	0.0	0.0113	0.0113	0.0	0.0	0.0023	0.0
0.0	0.3923	0.2152	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3923	0.0002
0.0	0.25	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.25	0.0
0.0	0.0426	0.0426	0.0091	0.0091	0.8507	0.0	0.0033	0.0	0.0	0.0	0.0426	0.0
0.0	0.0426	0.0426	0.0091	0.0091	0.8507	0.0	0.0033	0.0	0.0	0.0	0.0426	0.0
0.0	0.0426	0.0426	0.0091	0.0091	0.8507	0.0	0.0033	0.0	0.0	0.0	0.0426	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.5	0.0	0.0	0.0
0.0273	0.0023	0.0343	0.0	0.0	0.0	0.0433	0.4788	0.3662	0.0433	0.0	0.0023	0.0023
0.0307	0.0026	0.0387	0.0	0.0	0.0	0.0488	0.4127	0.4127	0.0488	0.0	0.0026	0.0026
0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.5	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1	0.0	0.0
0.0	0.333	0.1826	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4842	0.0002
0.0	0.0048	0.0027	0.0	0.0	0.0	0.0011	0.0016	0.0016	0.0011	0.0	0.0048	0.9822

Table 2.2: Confusion matrix estimated during the experiments performed on real data manually classified in advance. The actions in represented by rows and columns are following: `BRUTEFORCE`, `CONNECT_TO_C2`, `CONNECT_TO_HOST`, `DDOS_TO_HOST`, `DDOS_TO_SPECIFIC_SERVICE`, `DNS_REQUESTS`, `FINGERPRINTING`, `HORIZONTAL_PING_SCAN`, `HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE`, `PORT_SCAN`, `SEND_SPAM`, `UPLOAD_DATA_TO_C2`, `WEB_ATTACKS`. Note that order of rows and columns matches the alphabetical order of actions

The confusion matrix visualizes the performance of a classifier. Each column of the matrix represents the instances of the predicted class (classification result), while each row represents the instances of the actual class. The elements of the confusion matrix are probabilities of the classification P(predicted class|actual class) and the sum of rows in a column equals to 1. In our case predicted class represents observation o_k and actual class is the performed action a_j. For example, from the matrix C defined in Equation 2.4 we can easily see, that for this specific classifier (in this example we have used the classification of colors) the probability that red color will be classified as red is 0.9 and as blue is 0.1 and similarly for the blue color.

$$C = \begin{matrix} \\ Red \\ Blue \end{matrix} \begin{matrix} \begin{matrix} Red & Blue \end{matrix} \\ \begin{pmatrix} 0.9 & 0.2 \\ 0.1 & 0.8 \end{pmatrix} \end{matrix} \tag{2.4}$$

Finally, once we have estimated probabilities $P(o_k|a_j)$ and $P(a_j|d_i)$, we can compute the observation probability stated in Equation 2.2 using the Eq. 2.5. Note that the set of actions is extended with special action called `NOOP` representing the action when attacker performs no action or some action is classified as normal traffic. This represents a *false negative*.

$$\forall i \in \{1 \dots m\} P(o_k|d_i, a_j) = \begin{cases} P(a_j|d_i)P(o_k|a_j) & \text{when } k, j \in \{1 \dots n\} \\ 1 - P(a_j|d_i) & \text{when } k = 0, j \in \{1 \dots n\} \\ 0 & \text{when } k \in \{1 \dots n\}, j = 0 \\ 1 & \text{when } k, j = 0 \end{cases} \tag{2.5}$$

The first case in the definition represents the situation when attacker performs action a_j, the detection layer using aggregation function d_i detects this action as malicious and the classification phase labels this action as an observation o_k. The second case represents the situation when the

detection phase labels the action a_j as legitimate traffic. This case represents *false negative* error of the system. The third and fourth case represents the situation when attacker performs no action. In previous paragraphs, we have described the detection and classification process and we have stated the assumption that only the attacks are taken in account during the estimation of the observation probability $P(o_k|a_j, d_i)$. Therefore, when attacker performs no action, there are no observable flows. Consequently, their label (on the basis of this strict assumption) has to be always `NOOP`. The Formula 2.5 is rewritten into following matrix.

$$C_{d_i} = \begin{pmatrix} 1 & 1 - P(a_1|d_i) & 1 - P(a_2|d_i) & \dots & 1 - P(a_n|d_i) \\ 0 & & \vdots & & \\ 0 & P(o_k|a_1)P(a_1|d_i) & P(o_k|a_2)P(a_2|d_i) & \dots & P(o_k|a_n)P(a_n|d_i) \\ \vdots & & \vdots & & \\ 0 & & & & \end{pmatrix} \tag{2.6}$$

The submatrix containing the probabilities $P(o_k|a_j)P(a_j|d_i)$ represents the confusion matrix specifying the performance of the classification. The matrix 2.6 represents the joint properties of confusion matrix for both the detection and classification process. The verification of the results in the first column is trivial. For the sum of the rest of the columns following equations holds.

$$1 - P(a_j|d_i) + \sum_{k=1}^{n} P(o_k|a_j)P(a_j|d_i) = \tag{2.7}$$

$$1 - P(a_j|d_i) + P(a_j|d_i) \underbrace{\sum_{k=1}^{n} P(o_k|a_j)}_{=1} = 1 \tag{2.8}$$

Chapter 3

Game-Theoretic Model of the Attacker/Defender Interaction

3.1 Formal Game-Theoretic Model of the Attacker/Defender Interaction

We aim to use the game-theoretic models to improve the security of the adaptation process within a distributed, agent-based Intrusion Detection System (IDS). Adaptation, self-management and self-optimization techniques that are used inside an Intrusion Detection Systems (IDS) significantly improve their performance [46] (i.e. reduce the number of false alarms) in a highly dynamic environment, but are also a potential target for an informed and sophisticated attacker. When the adaptation techniques are deployed improperly, they can allow the attacker to reduce the system performance against one or more critical attacks. This chapter we discuss game theoretical models of adaptation processes inside an autonomic, self-optimizing Intrusion Detection System.

Our goal is first and foremost to analyze the risks related to opponent's manipulation of system internal state and configuration, performed in order to reduce its effectiveness. This addresses the existing concern with expected increase in malware (malicious software) sophistication - theoretical models for distributed learning in malware exist [29], and strategic manipulation of Intrusion Detection Systems by shaping of the input data has been demonstrated, albeit offline [47]. This behavior corresponds to wider context of targeted attacks on learning processes, studied in the fields of adversarial machine learning and adversarial classification [6]. To address this issue we have made several experiments with attacks to learning process of each of the detection methods. Results of these experiments can be found in Section A.

Therefore, if we want to introduce an environment-driven adaptation into an industrialized intrusion detection system, we need to determine what is the extent to which can the opponent misuse the adaptation functionality to reduce system's effectiveness, and we need to model the attacker as an informed, strategically behaved entity performing a targeted attack rather than a random threat. Specifically, we present:

- **Architectures** integrating the abstract game models into an IDS with self-monitoring capability, in order to simulate the worst case, optimally informed attacker. Such (hypothetical) attacker with full access to system parameters could dynamically identify the best strategy to play against the system. Optimizing the detection performance against the worst case

attacker protects the system from more realistic attacks based on long-term probing and adversarial machine learning approaches referenced above.

– **Experiment** answering the following crucial question: What is the cost of preventive IDS resistance to such attackers with access to internal state information and outputs of an IDS? In other words, we measure whether and by how much will the preventive randomized IDS reconfiguration against the "worst case", highly sophisticated attacks with insider access reduce its performance against the "standard", relatively unsophisticated attackers with no knowledge of IDS existence, nominal effectiveness and current internal state.

In the following, we conceptualize the relationship between the attacker and the defender from the general point of view.

3.1.1 IDS Game Definitions and Assumptions

The game model (and utility functions in particular) are based on [12], with additional inputs from the network administrators and actual IDS users and inspiration from earlier work [1, 2, 34]. The game model integrates the preferences and strategies of two players (attacker and defender). Their strategy sets are defined as a selection of IDS configurations for the defender and the selection of a particular attack type (e.g. buffer overflow, password, brute-force, scan...) for the attacker. The main difference of the utility functions from [12] is the relaxation of the requirement on the identical attacker gain/defender loss and the proportionality of associated costs (alarm processing, monitoring etc.) with the gain/loss value. We were able to relax this requirement in our model and implementation, as it was subject to critique from our clients: practitioners in the network administration field. In the following section we have kept the values identical in our experiments to comply with Chen [12].

The actual utility function values of both players depend principally on the sensitivity of the system using defender's strategies with respect to individual attacker's strategies, and the associated rate of false positives for each configuration. By our experience, these values wary widely with changing characteristics of the background traffic, and need to be estimated dynamically for each given game in a sequence, as we will present below.

The key dynamic parameters of the model are:

– $\alpha_{i,j}$ denotes the probability that the j-th attack strategy is detected by the IDS when the defender plays the i-th defense strategy.

– β_i denotes the probability that the i-th defender's strategy will result in a false positive (false alert) on a particular connection/flow/packet.

3.2 One Stage Game

In this section, we will use the simplest model available in the field of the game theory, a single stage game of two players. Thus we model the interaction between defender and attacker as sequence of single stage, two player, non-zero sum games, where the attack/defense actions of both players correspond to strategies in the game-theoretical model of their interaction.

3.2.1 Game Model

Many of the game-theoretical models used in the security domain use the Stackelberg form [61], where one of the players assumes leader's role (defender) and the other follows. In our case, this format is not appropriate: both players act without any knowledge of the actions performed by the other player. The attacker acts first, and the defender is unable to perceive the attacker's actions before committing to a particular strategy.

Each such game is defined as a three tuple:

$$G = (N, S, U) \tag{3.1}$$

- where N is a set of **players** denoted $N = \{d, a\}$, where player a is the attacker (column player) and player d is the defender (row player),
- S is a set of **strategies** available to players. In our case, where the strategies are disjunctive, we impose simply $S = \{d_1, ..., d_i, ..., d_m, a_1, ..., a_j, ..., a_n\}$, here the strategies d_i are those of the defender and the strategies a_i are available to the attacker, and
- U denotes **utility function** of the form: $U : S \times S \rightarrow \mathbb{R} \times \mathbb{R}$, or less formally: $U : d_i \times a_i \rightarrow (u_d, u_a)$. Utility function returns the game payoff of the defender u_d and the attacker u_a when these invoke the strategies d_i and a_i respectively.

The gameplay of this game type is very simple in our case: both players simultaneously select their strategies from the set S and the combination of these strategies determines the payoffs to attacker and defender, as defined by their respective utility functions. Note that due to the inherent nature of the IDS problem, the game is not a zero sum game. The solution concepts used to solve/analyze the game are **Max-Min** and **Nash** equilibria[1].

In a real system, we don't play a single game, but rather a sequence of the games described above, each corresponding to a particular time interval. The individual games in the sequence are differentiated by the dynamically evolving parameters of player's utility functions as defined below. As we are only considering the adaptation use-case, we consider the individual games in the sequence to be independent and we don't carry over any information between them. The game integration with IDS architecture is described in Section 3.2.2, where we also discuss other model considerations from the system perspective.

Strategies

The defender's **pure strategies** are defined as a selection of one system configuration from a finite number of available configurations, with mixed strategies defined on this support.

The attacker's pure strategies are defined even more easily. Each attacker's strategy is defined by performing one attack such as horizontal scan, vertical scan, host fingerprinting, buffer overflow, denial of service and others. Mixed strategies are defined similarly to those of the defender, as a probability distribution on the support of attack actions. In practice, the attackers also execute their strategies in a particular, logical orderings (plans), but the identification and use of this behavior is outside of the scope of this paper.

[1] The main practical difference is the need for opponent modeling - should the players use the concept of Nash equilibria, they need either to know each other's utility function or they need to estimate the shape of the utility function from other agent's behavior [29]. This knowledge is not necessary in the max-min approach, where the players rely only on the knowledge of their own utility functions.

Utility Functions

The form of the utility functions determines the characteristics of the game – if the game is a **zero sum game**, i.e. the sum of utilities of all players is constant over any combination of played strategies, it is relatively easy to identify stable Nash equilibria. However, in our case, the game is not zero sum. This is a natural corollary of the criminal character of the activities [7] – crime can be commonly defined as an activity that redistributes the utility between the players at the expense of the overall utility reduction.

The utility functions in our game are relatively complex, as they need to reflect the complexity of the problem. The parameters of both utility functions are:

- $\alpha_{i,j}$ denotes the probability that the attack strategy a_j is detected when the defender selects the defense strategy d_i. Intuitively, in our case, it estimates the probability of the given detection strategy (i.e. aggregation function) being able to successfully raise an alarm upon the occurrence of an attack from the class corresponding to a_j. It depends on current status of detection agents and the background traffic.
- β_i denotes the probability that a given detection strategy, combined with current system state and background traffic, will result in a false positive. Note that this element only influences defender's payoffs, and its manipulation can be used by the attacker to launch denial-of-service attacks on detection mechanisms [39].
- γ_j denotes the probability of attack success. It discounts the value of undetected breach from both the attackers and defenders standpoint, and typically features low values.
- V denotes the background traffic volume in the game time (average, aggregate or other appropriate value according to the underlying IDS system nature) that is used to estimate the number of false positives together with the parameter β_i.
- $P_d(a_j)$ denotes the defender's payoff/loss on attack success. It is most often a negative value, except for multi-tier honeypot systems or very particular situations where the defender can gain knowledge from the attacker bypassing the IDS. The loss can be relatively low in case of exploratory activities (scan, fingerprinting), but is relatively high when the attacker actually breaches a system.[2]
- $P_a(a_j)$ denotes the utility the attacker expects to receives upon successful realization of given attack action from the attack class corresponding to strategy a_j.
- $D_a(a_j)$ denotes the attacker's payoff/loss on detection, which is typically a negative value. The value of this parameter can vary widely, as it can be very high (in absolute value) for last stages of elaborate attacks executed inside defender's perimeter, or can be almost zero in case of Internet attacks.
- $D_d(a_j)$ denotes the defender's payoff for attack detection. This parameter value is the main cause for the game not being a zero sum game in a general case, as the payoff is typically zero in enterprise or Internet settings following the similar reasoning as in the $P_d(a_j)$ case – damages from the attacking party are almost impossible to seek (not even considering the problems related to the root attacker identification and burden of proof).
- $C_a(a_j)$ denotes the cost of the attack execution on the part of attacker. Typically very low for Internet-originating attacks.

[2]The attack tree-based methodology introduced in [45] allows the distribution of this ultimate utility between the pre-condition actions, effectively motivating the system to concentrate on all relevant stages of the attack (equivalent to attack classes a_j) during the game.

- C_{TP} - denotes the (average) cost of processing of each detected incident (true positive) for the defender.
- C_{FP} - denotes the average cost of a false alarm for the defender, used in conjunction with β and V to estimate the false positive cost.
- C_M - denotes the fixed cost of monitoring infrastructure, independent on attack or traffic intensity.

The utility function of the defender has three principal components: the first term deals with successfully detected attacks, the second term represents the loss associated with undetected attacks and the third term describes the overhead of the monitoring, which consists of the false positive costs and the fixed cost of monitoring.

Individual utility functions are defined as follows. Defender's utility is:

$$\begin{aligned} u_d(d_j, a_i) = \alpha_{i,j}(D_d(a_j) - C_{TP}) + (1 - \alpha_{i,j})\gamma_j P_d(a_j) \\ -\beta_i V C_{FP} - C_M \end{aligned} \tag{3.2}$$

Attacker's utility can be described as:

$$u_a(d_j, a_i) = \alpha_{i,j} D_a(a_j) + (1 - \alpha_{i,j})\gamma_j P_a(a_j) - C_a(a_j) \tag{3.3}$$

Utility function terms. The first situation that we represent corresponds to attack detection. The term in the defender's utility function describing this situation (without the fixed costs of monitoring and false positives, which will be discussed below) is: $\alpha_{i,j}(D_d(a_j) - C_{TP})$. We can see that the defender may get some payoff from attack detection, but globally, the value of the term $D_d(a_j)$ would be zero or negative due to the reinstallation and recovery costs. The term C_{TP} represents the immediate cost of incident detection, investigation and processing.

On the attacker's side, this situation is described by the term: $\alpha_{i,j} D_a(a_j)$. We can see that the loss of the attacker depends almost entirely on the impact the detection has on attacker's plans – the value can be relatively high in the last stages of complex attack plans deep within the protected perimeter, but is next to zero for malware propagation on Internet due to the lack of effective enforcement.

The second term of the utility function covers the situation when the attacks are not detected. In the defender's case, the term is:$(1 - \alpha_{i,j})\gamma_j P_d(a_j)$. The first factor corresponds to non-detection probability, while the factor γ_j describes the likelihood of attack/exploit success, which then amortizes the value of the successful execution $P_d(a_j)$. The impact of the factor γ_j is crucial. When there is an attack, the actual *defender's optimum in most situations is that the attack is both undetected and unsuccessful*. Reasoning behind this analysis is straightforward: The term $D_d(a_j)$ is typically zero or negative[3], the term $-C_{TP}$ is also negative and the best strategy is therefore to avoid detection of attacks with low loss/likelihood product $\gamma_j P_d(a_j)$.

[3]The only situations where this term is actually globally positive are those where an efficient counter-attack mechanism (in tactical/military problems) or intelligence-processing mechanism allows the defender to counter-attack the attacker's resources or to deduce attacker's goals, plans or at least intentions. From the other side of the problem, the attacker needs to structure its actions in such a way, that their eventual compromise would not give away disproportional high volume of information about its goals or resources. This consideration is integrated in the value of the term $D_a(a_j)$.

From the attacker's perspective, the second term is a straightforward amortization of success payoff by success likelihood and non-detection probability. Failure to exploit (with probability $1 - \gamma_j$) is also preferable to detection for the attacker, but only by the slight margin of the term $D_a(a_j)$, which is typically low for external attacks: $(1 - \alpha_{i,j})\gamma_j P_a(a_j)$. The conclusion that some (i.e. unsuccessful) attacks are better left undetected may seem surprising, but it actually corresponds to very natural equilibria, due to the costs associated with any detected attacks. The problem in this case is therefore how to optimize the sensitivity of the intrusion detection system, so that it will only detect the relevant threats. We attempt not only to remove the false positives, but also discount the value of true positives with little relevance to the actual system. This behavior ensures more effective monitoring with little or no impact on security.

The last component of both utility functions captures the costs related to cyber-attack or defense. Attacker's side lost utility can be trivially described as the cost associated with attack performance:$-C_a(a_j)$. The defender's utilities depend on two principal factors: cost of the monitoring infrastructure and the cost of the processing of false positives, which can be significant for real world systems: $-\beta_i V C_{FP} - C_M$. The first of the two components estimates the number of false positives ($\beta_i V$) and the total cost of their assessment ($\beta_i V C_{FP}$), while the second term captures the fixed cost of monitoring, such as the infrastructure cost and fixed operation costs.

The size of these two terms is non-negligible – the number of false positives can rival or outnumber the number of real incidents in open networks, and false positives would typically significantly outnumber the true positives on internal, well-managed networks. These two terms are also the main reason why the IDS game is not a zero sum game, as they introduce a fundamental non-efficiency into the system.

3.2.2 Architecture for Game-Theoretic Online IDS Reconfiguration

In this key section, we will describe the integration of the game-theoretical model with the adaptation process of a particular IDS. This integration consists of several steps: dynamic parameter estimation in the system, game definition, game solution and integration of results back into the system.

Game Integration Techniques

There are two existing integration options addressing the problems from the opposite sides of the spectrum:

- *Off-line integration*, when the game is defined in design time, solved analytically, using priori knowledge about expected impacts and success likelihood of the attacks, and the system parameters are fixed to resulting strategies according to game results[57]. This is the most traditional way of using the game theoretical methods, as their use ensures that the system parameters are set to force the adversary into the selection of less damaging (or more rational) strategies. The advantage of this approach is relatively easy solution identification and low technical difficulty, but the disadvantage is the fact that the game solutions identify the behavior that is advantageous on average, and do not reflect the dynamic changes of assumptions, threat characteristics and background traffic. This is sufficient for systems deployed in stable environments, but most IDS need to cope with dynamic environments, where the background traffic and other factors change frequently. In such environments, the static (albeit mixed) strategies covering such wide range of environments perform poorly.

– *Direct on-line integration*, when the game uses presumed adversary actions in the observed network traffic to define the game is the opposite approach. The game is being defined by the actual actions of real-world attackers executed against the monitored system. This approach addresses the problem with game definition relevance by using the actual attacks and traffic background to define the game at runtime. The game is then solved as an optimization problem, but with several drawbacks. Direct interaction between the adversary and the adaptation mechanism makes the system potentially vulnerable to attacks on machine learning and adaptation algorithms [6], making the whole IDS potentially less secure than without the use of game-theory driven adaptation. Motivated attacker can easily mislead the IDS by insertion of a sequence of attacks that are orthogonal to its actual plan, and that would make the IDS less sensitive w.r.t the actually dangerous attacks.

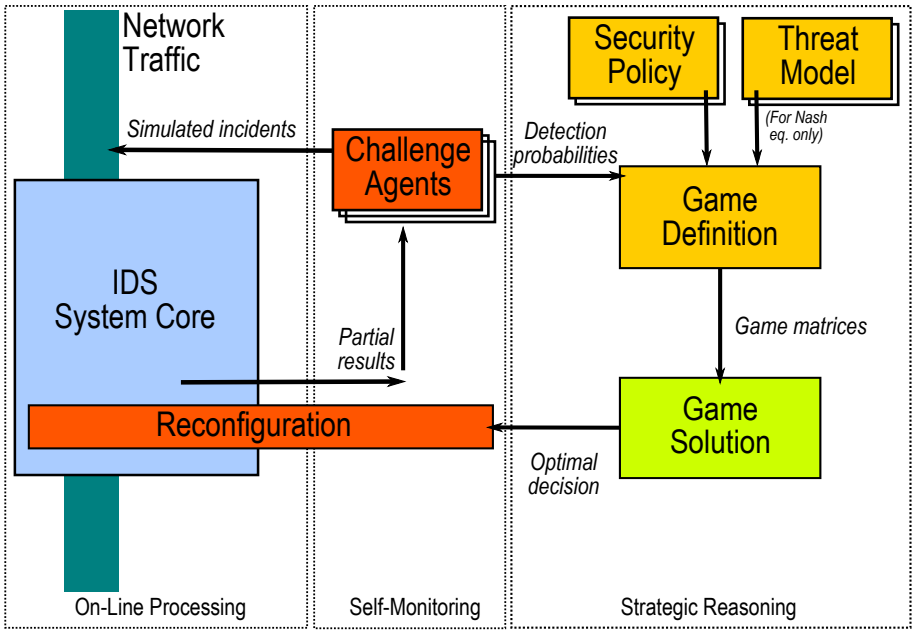


Figure 3.1: Indirect online variant of integration of the game with IDS.

Our approach, named *indirect online integration* [46] combines the above approaches and provides interesting security properties desirable for real-world deployment. The solution uses the concept of challenges to mix a controlled sample of legitimate and adversarial behavior with actually observed network traffic and is a compromise between the above approaches (see Fig. 3.1). In this case, the real traffic background (including any possible attacks) is used in conjunction with simulated hypothetical attacks within the system. These attacks are then mixed with the real traffic on IDS input and the system response to them is used as an input for game definition. The major advantage is higher robustness w.r.t strategic attacks on adaptation algorithms, and lower system configuration predictability by the adversary, as the simulation runs inside the system itself and its results cannot be easily predicted by the attacker.

This approach offers the optimal mix of situation awareness and security against engineered inputs. In this case, we actually play against an abstract opponent model inside the system, and expect that the moves that are effective against this opponent will be as effective against the real attacks. The advantage of this approach is not only in its security, but also in better model characteristics in terms of strategy space coverage (less frequent, but critical attacks can be covered), robustness and relevance – the abstract game can represent the attacks and utility combinations that would be obvious only for insider attackers.

Indirect Online Integration

The use of the indirect online integration in practice requires a division of the covered time interval into sub-intervals defining each single game is a sequence. The length of such interval depends on the IDS technology used, line speed, hardware performance and other factors – it can vary between few seconds for pattern-matching packet filters to few minutes/hour for statistical anomaly detectors.

During each interval t, the system measures/estimates the values of parameters (in particular the detection probabilities $\alpha_{i,j}$ and the false positive probabilities β_i). For the reasons listed above, we suggest the use of challenge-based parameter estimation [46], which relies on insertion of known instances of legitimate or malicious behavior into the background, unclassified traffic. We measure the system response to these challenges, drawn from the realistic attack classes, and use them to estimate the system response to all real-world samples from the same classes. In practice, we will define one class for each broadly defined attack/legitimate traffic type and measure the difference between the system response to legitimate traffic and to various classes of malicious traffic. The adaptation process will then assess the statistical properties of the response and use them to estimate the detection probabilities $\alpha_{i,j}$ and the expected ratio of false positives β_i for given defender's strategy i. It is worth noting that this method is based on the assumption that the response of the detection method used in the IDS against members of each class is consistent and that the anomaly scores of the class members are distributed according to normal distribution. This assumption has been verified in our past work, and can be ensured as the attack class definition is under the full control of game designer – if the response to one of the classes is for example multimodal, it can be easily split into two separate classes.

The game definition ordering with respect to each time interval also depends on the type of the underlying IDS. The CognitiveOne system introduced in the Appendix is a NetFlow based collective anomaly detector, and therefore processes the data in well-defined and regularly produced batches rather than in real time - this means that the game is actually defined **after** the data has been recorded. In case of traditional pattern matching IDS that needs to operate on wire speed, the game needs to be defined and solved **beforehand**, so that the strategies can be applied directly to each processed packet, flow or connection. In practice, this means that the systems solving the game after the interval t on which the solution is applied have precise parameter estimations for each particular interval, while the wire-speed systems would apply the t-th game results to the interval $t + 1$.[4]

In both cases, once the system obtains the game definition and solves it, it can directly apply the results back into the system configuration and use then on current or next time interval.

Game Strategies for Real World IDS

The existing self-monitoring capabilities of the CAMNEP system are essential for online empirical estimation of the key utility function coefficients $\alpha_{i,j}$ and β_i, as their values typically evolve throughout the day.

The CAMNEP system is based on a self-organized, multi-level collaboration of detection agents, each of them maintaining an different model of traffic normality/anomaly. The agents share the

[4]The slight delay of application is unlikely to cause a problem, as suggested by our experimental results. The system using the parameters weighted over 5 last intervals performed comparably with the one using only the precise values for the specific interval.

anomaly estimates at various stages of processing and once they have reached their partial conclusions (anomaly scores for each network flow/connection), the system needs to aggregate these opinions together. At this stage, it is important to notice that the performance of individual detection agents and their combinations varies with background traffic and attack types.

The **defender strategies** in CAMNEP are instantiated as specific aggregation functions used to integrate the opinions of detection agents in the system. Defender's strategy selection is thus technically straightforward, as it only picks one particular aggregation operator that aggregates diverse expert opinions with particular weights or methods. In our experimental system, there were 30 operators aggregating the opinions of 6 detection agents in total.

3.2.3 Experimental Evaluation

For the single-stage game we verify the capability of the system to detect a real-world attack and compare various game-solving method with the trust used in the original CAMNEP system. To measure the quality of detection, we have to define new criteria, as the one used for challenge insertion evaluation cannot be applied to external attacks for the lack of data. Therefore, we use the criteria:

$$\mathcal{E} = \frac{\bar{\theta}_{all} - \bar{\theta}_x}{\sigma_{all}} \tag{3.4}$$

where the $\bar{\theta}_{all}$ is the arithmetic mean of the anomaly values of the whole observed traffic, $\bar{\theta}_x$ is the arithmetic mean of the flow anomaly values of the measured attacks described above and σ_{all} is the standard deviation of the whole traffic anomaly values. Higher values are better, as they ensure better separation of the anomalous traffic. Table 3.1 shows how the individual solution concepts can select the strategies that separate the individual attack types from the legitimate traffic. We can clearly see that some of the attacks are more difficult to detect (Horizontal UDP scan), and result in negative values of the criteria, as the traffic is less anomalous than the average flow. The aggregate results shown in the last two lines confirm the assertion (*iii*), as the results of all three methods are closer to each other, and two GT concepts outperform Trust in case of longer time horizon (MM5, NE5 and Trust5 columns). The assertion (*iv*), which postulates that the values with longer time horizon shall outperform the short-term horizon columns, does not hold, as it is only true for the Nash equilibria. The assertion (*v*), the key hypothesis that this paper tries to disprove, states that the game theoretical methods may underperform the trust-based solutions in the situations when the opponent does not behave strategically. The difference in performance is due to the randomization and inefficiencies related to second-order strategic behavior. This (rather pessimistic) hypothesis does not hold. While the game theoretical methods score slightly less, the actual lost utility is surprisingly low, lower than the difference due to the use of longer history in directly optimizing trust function. This is also evident on Fig 3.2, where we can see the performance of different solution concepts on the incident inserted over time (The function comb-like pattern is caused by the fact that not all time period contain the real attacks). The figure shows that all solution concepts score very close to each other, and that the penalty related to their use can be neglected.

Next we have made comparisons of trust-based adaptation and single stage game adaptation when using only random generated challenges, planned challenges or both together. In these experiments we are measuring so called score. The score represents the distance between malicious and legitimate parts of the testing flows sets. In more detail the score is defined as:

$$score = \frac{sgn(\bar{\theta}_{mal} - \bar{\theta}_{leg})(\bar{\theta}_{mal} - \bar{\theta}_{leg})}{\sigma_{leg} + \sigma_{mal}} \tag{3.5}$$

Table 3.1: Average value of $\mathcal{E}$ over all datasets for each solution concept with respect to attack classes.

	MM1	MM5	NE1	NE5	Trust1	Trust5	Avg. fct.	Best fct.
SSH brute force	4.141	4.153	4.115	4.215	4.318	4.057	3.862	5.527
Vert. TCP scan with OS detection	0.840	0.869	0.783	0.852	0.885	0.908	0.780	-1.245
Vert. UDP scan	2.200	2.191	2.112	2.177	2.252	2.306	2.030	0.262
Horiz. TCP scan for SSH	1.158	1.305	0.471	1.191	0.977	0.561	0.639	2.560
Horiz. UDP scan for DNS	-1.735	-1.376	-1.590	-1.375	-1.416	-1.749	-1.464	0.852
Horiz. ICMP ping scan	4.787	4.107	4.296	4.282	4.884	4.626	4.185	8.775
Horiz. TCP scan for several services	3.069	2.970	3.064	2.980	2.965	3.031	3.191	4.015
Sum	14.459	14.218	13.251	14.323	14.866	13.740	13.222	20.745
Average	2.066	2.031	1.893	2.046	2.124	1.963	1.889	2.964

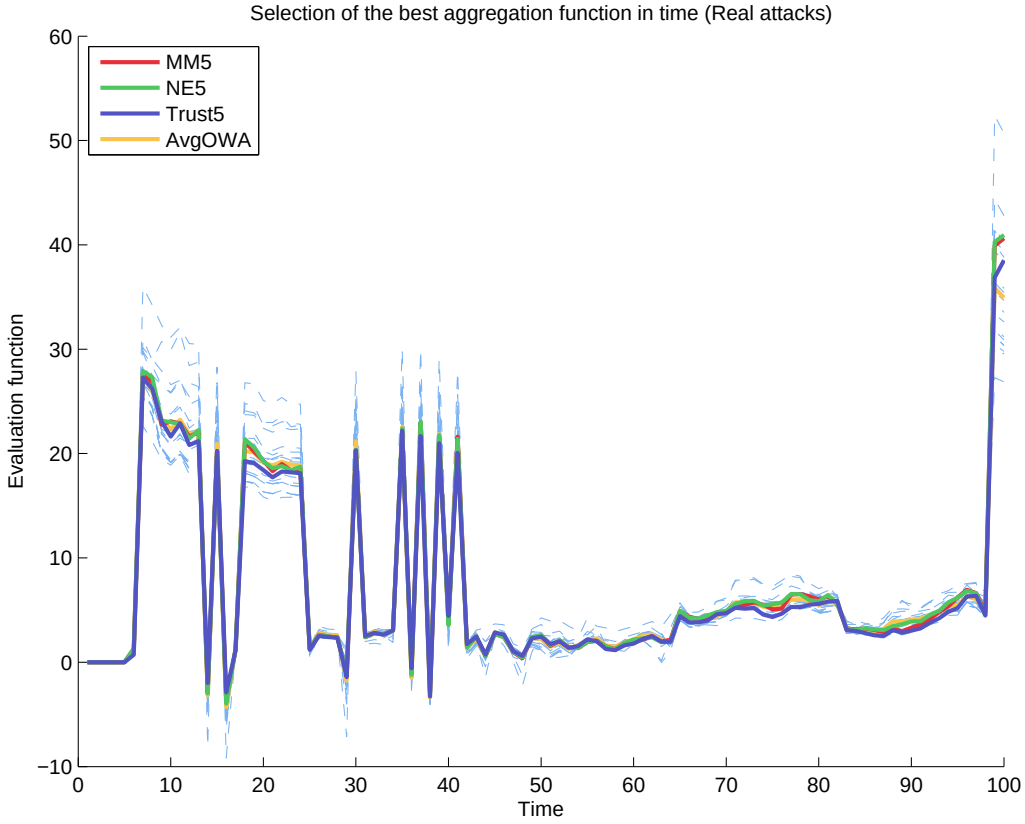


Figure 3.2: Performance of solution on real attacks over time, using the criteria 3.4.

where the $\bar{\theta}_{mal}$ is the arithmetic mean of the trust values of the malicious flows, that were manually labeled in the traffic, the $\bar{\theta}_{leg}$ denotes the arithmetic mean of the trust values of the legitimate flows and σ_{leg}, σ_{mal} denotes the sigma of the legitimate and malicious flows respectively.

Figure 3.4 shows the comparison of the score values of the presented challenge insertion processes when using trust-based adaptation. From the figure can be seen that using only planned challenges approach we can obtain the best results. The reason for the combination of both available insertion processes to perform so poorly is the volume of the simulated traffic. When using both approaches together, the volume of the inserted flows is more than twice greater than the volume of the real network traffic.

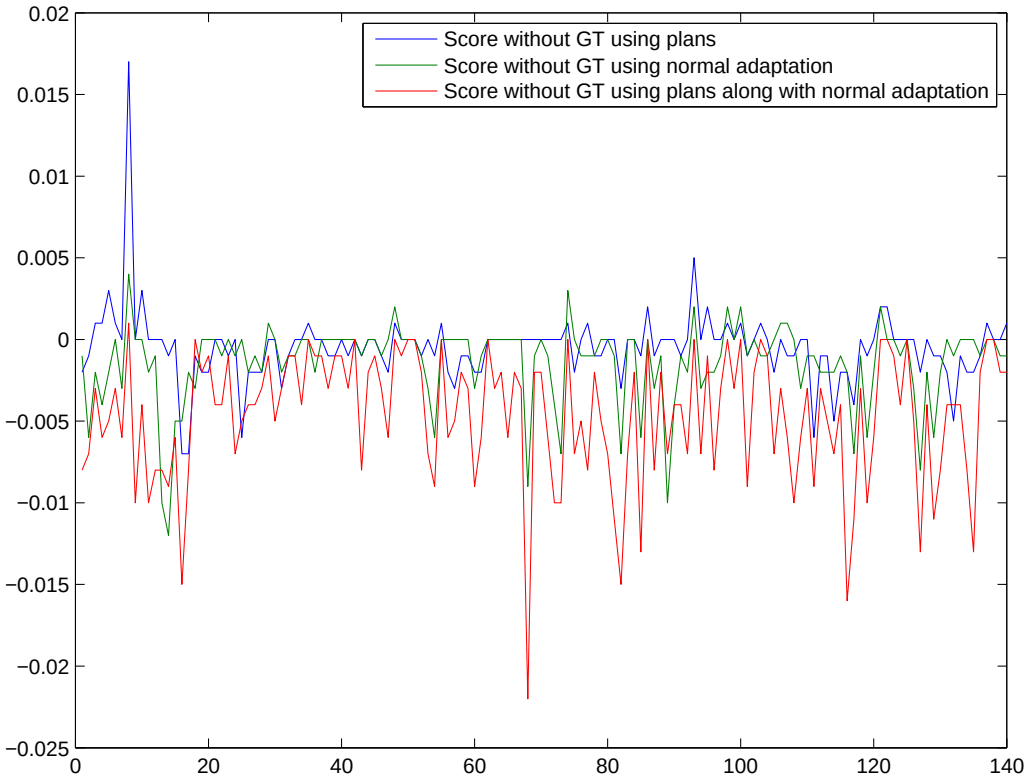


Figure 3.3: Comparison of the performance of the system that uses one trust-based adaptation process with only planned challenges, challenges from the trust-based adaptation process and combination of both challenge insertion processes.

Figure 3.4 similarly to the Figure 3.4 compares performance of the challenge insertion approaches when using one stage game adaptation. caused by the fact that contrary to the trust-based adaptation the one stage game does not maintain any history and thus is not leveled off by the history values.

3.3 Adversarial Plan Recognition Game

In the reminder of this chapter, we will develop a more complex game model. It takes into account the progression of the game in time and the gradual computation of the defender's belief about the plan executed by the attacker. The problem solved by the game is in principle a plan recognition problem.

Intention and plan recognition is an important capability of an intelligent system. It enables coordination with other agents in situations where explicit communication is not possible or desirable. The techniques for plan recognition has been studied in the context of unobtrusive intelligent assistance to people or coordination of robotic systems. However, most of the approaches to plan recognition assume that the agent, plan of which is being recognized, is either indifferent about the recognition process, or actively cooperates and gives hints that make the recognition process easier.

On the other hand, the problem of recognizing a plan of an agent with adversarial utilities is a much

Figure 3.4: Comparison of the performance of the system that uses one stage game-based adaptation process with only planned challenges, challenges from the trust-based adaptation process and combination of both challenge insertion processes.

less studied problem. We term this problem as *adversarial plan recognition*, and we understand it to be a problem of recognizing a plan or intentions of an adversary that is aware of the presence of the recognition process and actively tries to avoid it while pursuing some other tasks. The adversary may choose actions that are hard to detect, or use deception techniques to make the recognizer waste its resources. The adversarial plan recognition is widely applicable in various domains, in which longer and repeated mutual interaction of adversarial agents is common. Examples can be found in warfare scenarios, scenarios from different security domains, or computer networks. The latter is the main use case we use for experimental validation of the proposed techniques.

The competitive characteristics of the goals of interacting agents in the problem of adversarial plan recognition directly lead to the usage of game-theoretic approach for solving the problem. Therefore, we define the problem of the adversarial plan recognition as a two-player zero-sum extensive-form game between the attacker and the defender. None of the players can directly observe the actions taken by the other player; however, we assume that the defender receives some noisy probabilistic observations from the environment about the actions of the attacker. In our approach we show that thanks to the special structure of the game, the size of the game representation can be substantially reduced. Moreover, if the actions of the defender do not have any preconditions and the probability distributions of the observations depend only on the last action of the defender, the best response (given the strategy of the attacker) can be computed quickly. We utilize both of these findings and introduce a novel algorithm for the problem of adversarial plan recognition, that is based on the current state-of-the-art game-tree search algorithms, Monte-Carlo Tree Search (MCTS), and that good approximates the optimal solution under real-world

time constraints.

3.3.1 Assumptions

Following the game-theoretic methods we define the adversarial recognition problem to be a game between the attacker trying to achieve some of its goals unobserved, and the defender using different sensors (action classifier) to detect the plan of the attacker. Such a game can be in general computationally intractable due to the presence of an uncertainty and possibly large strategy space of both players. Therefore, we pose following assumptions for the game model we use in our approach:

- **Both players, the attacker and the defender, are rational.** We assume that the players choose actions that optimize their utility based on the information available to them.
- **All actions of the attacker have equal length.** We assume that the game is played in discrete stages and the attacker executes exactly one action in each stage.
- **The defender can use only one classifier at a time.** The resource limitation of the defender does not allow execution of more than one classifier at a time. Note, that even an assembly of classifiers can be seen as a single classifier.
- **Single attacker executes a single plan.** We assume that the game is played against a single attacker that executes only one action at a time.
- **Both players know the full plan library of the attacker.** Both players know the set of all plans that can possibly be used by the attacker.
- **The quality of the classifiers does not change during the game.** The quality of the classifiers defined by their confusion matrices stays the same in all stages of the game.
- **The available classifiers and their quality are known to both players.** Both players know how many classifiers are used by the defender as well as their exact confusion matrices.

Of course, the assumptions can come at the cost of limited expressiveness of the proposed game model; hence, we discuss in Section 3.7.2 whether these assumptions are satisfied in the domain of computer network security and which of them can be relaxed.

3.3.2 Extensive Form Games

Let us define an imperfect-information extensive-form game (EFG) that we use further. The game is a tuple $G = (N, A_a, A_d, A_n, H, Z, \rho, \sigma, u, I, \pi)$, where:

- N is a set of players; we assume there are two actual players – the attacker and the defender. The third player – termed Nature – is modeling the stochastic nature of the game in terms of uncertainty in the effects of the actions
- A_a (A_d respectively) is a set of actions available to the attacker (or to the defender respectively)

- A_n is a set of actions for the Nature player that reflects the uncertainty in the environment – the actions of the Nature player represent the responses of the system in terms of detecting or not detecting an anomalous event and its correct classification; for simplicity we assume $A_n = A_a$
- H is a set of nonterminal choice nodes;
- Z is a set of terminal nodes, disjoint from H;
- $\rho : H \to N$ is the player function, which assigns to each nonterminal node a player $i \in N$ who chooses an action at that node;
- $\sigma : H \times \{A_a \cup A_d \cup A_n\} \to H \cup Z$ is the successor function, which maps a choice node and an action to a new choice node or terminal node such that for all $h_1, h_2 \in H$ and $a_1, a_2 \in A$, if $\sigma(h_1, a_1) = \sigma(h_2, a_2)$ then $h_1 = h_2$ and $a_1 = a_2$;

  we denote $\sigma(h) \subseteq H$ to be a set of all successors of a node $h \in H$;

  we denote $\sigma(H', a) \subseteq H$ to be all successors of nodes $H' \subseteq H$ such that $\sigma(H', a) = \bigcup_{h \in H'} \sigma(h, a)$;

  finally, we use index of the player $\rho(h) = i$ to emphasize which player is currently making a decision – $\sigma_i(h)$
- $u = (u_a, u_d)$, where $u_i : Z \to R$ is a real-valued utility function for player $i \in N$ on the terminal nodes Z.
- $I = (I_a, I_d)$ is a set of information sets for the attacker and the defender, where $I_i = (I_{i,1}, \ldots, I_{i,k_i})$ is a partition of $\{h \in H : \rho(h) = i\}$ for the attacker or the defender i, and $\rho(h_1) = \rho(h_2)$ whenever there exists a j for which $h_1 \in I_{i,j}$ and $h_2 \in I_{i,j}$.
- $\pi : H \times \sigma(H) \to [0, 1]$ is a probability distribution on the successors of the choice nodes of the Nature player $\sigma_n(h)$ (i.e. $\rho(h) =$ Nature). The choice nodes of the Nature player act as chance nodes.

3.3.3 Adversarial Plan Recognition Game

The *adversarial plan recognition game* (APRG) is an imperfect-information zero-sum extensive-form game between the attacker and the defender.

Actions Any sequence of the actions of the attacker forms so called *attack plan*. Since many of the plans of the attacker begins with the similar sequence of actions, the set of all possible attack plans of the attacker forms an *attack tree*, where the edges correspond to the different actions of the attacker. Utility values for both players are assigned to each leaf of the attack tree, and the values represent the pay-off (the penalty) for the attacker (the defender) in the case that the attacker completes its plan according to the path from the root of the tree to the leaf unobserved.

The actions of the defender represent a selection of one from a fixed set of classifiers $(d_1 \ldots d_m)$. Each of the classifiers has a different quality of correctly detecting various attacker's actions. For each classifier, there is a known probability of detecting some of the actions given the attacker's real action. We call the detection of an action by a classifier an *observation*. Now, for a classifier d_j, the probability of detecting an observation o_i given attacker's action a_j is denoted as $P_j(o_i|a_i)$. The probabilities are given by a normalized confusion matrices, such as the one in Figure 3.5. In this example, the classifier d_1 is reasonably good in detecting action a_2 and a_3. The classifier d_2 is very good at detecting action a_1.

d_1

	a_1	a_2	a_3
o_1	0.5	0.1	0.1
o_2	0.2	0.7	0.1
o_3	0.3	0.2	0.8

d_2

	a_1	a_2	a_3
o_1	0.9	0.3	0.2
o_2	0.1	0.3	0.4
a_3	0.0	0.4	0.4

...

Figure 3.5: Example confusion matrix.

We assume that the game progresses in simultaneous moves. In each move, the attacker chooses a next action; the defender selects a single classifier. We assume that the defender's resource limitations do not allow using more than single classifier at a time. Then, nature selects the observation based on the confusion matrix and the next move is started.

Information Neither of the players can directly observe the actions of the other player. The defender can observe its own actions (selection of the classifiers) and the observations generated by the classifier. The attacker can observe only its own actions.

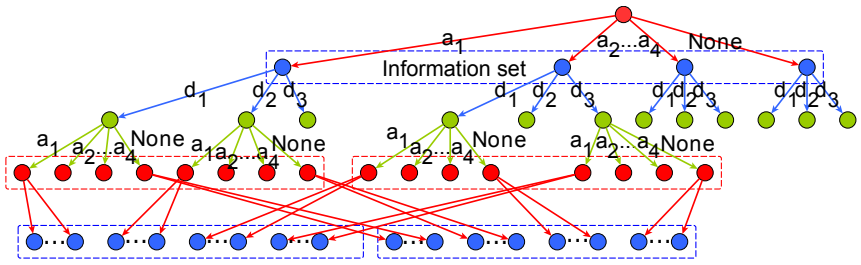


Figure 3.6: Portion of the extensive-form represenation of the game. The leafs in the picture are not actual leafs in the game, the game continues for large (possibly infinite) number of steps.

Part of the complete extensive-form tree of the game is in Figure 3.6. The tree of the game can be arbitrarily deep. The longest branch is three times longer than the length of the longest attack plan. Hence, the leaves in the figure are just root nodes of subtrees omitted for clarity. Three plies of nodes are periodically repeating in the tree. The attacker's decision nodes (red), the defender's decision nodes (blue) and the observation nodes (green). The attacker's decision nodes must respect the attack tree; hence, the available actions in one stage of the game are based on the actions performed by the attacker in the previous stages. The defender's decision node contains always the same set of actions corresponding to the different classifiers. The observation nodes are chance nodes with probability distribution based on the classifier quality matrices and the preceding actions of the attacker and the defender. The structure of the information sets is consistent with the assumptions stated above as well as with the intuition of simultaneous moves of the players. We refer to the three plies of different players starting with the attacker's decision as a *move*.

Utilities The utility function of the game captures the aims of the players described in the adversarial plan recognition problem. The utility value for the attacker therefore directly depends on the payoff g for the executed attack plan from the attack tree. Moreover, the value is discounted according to the number of times an action from the executed plan has been correctly observed:

$$u_A(a_1 \dots a_h, d_1 \dots d_h, o_1 \dots o_h) = \frac{g(a_1 \dots a_h)}{1 + \sum_{i \in \{1 \dots h\}; o_i = a_i} 1}$$

In the following we assume the game to be zero-sum, which is a natural assumption in the adversarial plan recognition; hence, this value is maximized by the attacker and minimized by the defender. However, as the presented methods in our approach are mostly independent of exact utility definition, this assumption does not have any significant impact on the complexity of the problem.

Solution Concept While selecting appropriate solution concept for the adversarial plan recognition game (APRG), we need to consider the aspect adversarial plan recognition problem. From the game-theoretic viewpoint, the most fundamental solution concept for imperfect-information extensive-form games is the Sequential Equilibrium, a refinement of the Nash Equilibrium concept and an imperfect-information-equivalent of the known subgame-perfect Nash equilibrium used in perfect-information extensive-form games [51]. This solution concept is suitable also for our problem. In the case of perfectly rational opponent, a Sequential Equilibrium describes the optimal strategy that guarantees the smallest possible loss for the defender and the highest possible gain for the attacker in the long run. The solution of the APRG will most likely be in the form of mixed strategies[5], since a deterministic selection of some action of the attacker would allow the defender to use the classifier with the best chance of detecting the performed action.

3.4 Solving Imperfect Information Extensive Form Games

Generally speaking, the adversarial plan recognition game (APRG) belongs to the class of extensive-form imperfect-information non-zero-sum games (II-EFG), for solving which there is a number of existing algorithms. In this section we thus give a brief description of these algorithms and discuss their applicability for the APRG.

3.4.1 Mathematical Programming

The exact algorithms for solving II-EFG are generally based on formulating the problem of finding the desired equilibrium (usually Nash Equilibrium, or its refinement Sequence Equilibrium) in the form of mathematical programming. Two basic options for computing Nash equilibria are summarized in [51]. The more efficient one is based on sequence-form representation of EFG. The sequence-form of EFG consists of (1) a set of all performable sequences of actions for each of the players, (2) a function that assigns payoffs to each combination of sequences for all players and (3) a set of linear constraints. In our game, if l is the look-ahead we use for playing the game, the size of the sequence form of our game is $O(|A_a|^l + |A_d|^l + |A_n|^l)$. The mathematical program for solving the sequence form game is a Linear Complementary Program of size polynomial in the size of the sequence-form of the game. The best known general algorithm for solving these programs is exponential in the size of the program. As a result, it is not likely that this algorithm will be usable for realistically large games. The situation is a simpler for zero-sum games, for which NE solution can be obtained by LP. However, the size of the game is most likely still prohibitive.

3.4.2 Counter-factual Regret Minimization

A little more practical algorithm for solving zero-sum II-EFG was developed by Zinkevich [62]. The counter-factual regret minimization (CFR) was used for creating a poker agent that was able

[5]Mixed strategy is a probability distribution among the actions of the player in each stage of the game since we have a game with the perfect recall.

to reach human expert level in simpler variants of the game.

The algorithm starts with a random strategy for each of the players and uses fictitious play to update the strategies. In each iteration, the strategy of one player is fixed and the strategy of the other player is updated in order to minimize the overall regret of the agent felt when playing against all the opponent strategies from the previous iterations. The strategy of the other players is adapted the same way in the following iteration.

This algorithm is guaranteed to eventually converge to a Nash equilibrium of the game and during the process, it can output a bound on the quality of the solution. However, the speed of convergence has been shown to be acceptable only experimentally. The original variant of the algorithm requires traversing whole game tree in each of the iteration and it took two weeks to compute competitive Poker strategy. A more advanced Monte-Carlo (MC) sampling version of the algorithm has been introduced[33], and might be considered for solving our game. However, the result is not likely to be positive.

The main reason why CFR algorithm has been successful in solving Poker is that a player in Poker has relatively small number of possible information sets. Most of the information (opponent's cards, cards in the pack) is unknown all the time. The number of information sets is much larger in our game. The players can observe their own actions as well as the actions of player Nature. The number of information sets of one player is $O\left((|A_a| + |A_n|)^l\right)$ and all of them must have a permanent record stored during the computation. Even the MC variant of the algorithm has to iterate over the whole tree in the first iteration. The size of the tree (and the worst case computational complexity of one iteration) is $O\left((|A_a| + |A_d| + |A_n|)^l\right)$.

3.4.3 Information Set Search

Information set search (ISS) [37] is a heuristic method for playing II-EFG that has been successful for example in Kriegspiel (imperfect information variant of Chess). However, it is a heuristic algorithm that does not provide any bounds on the quality of the produced solution. The main idea of the algorithm is substantial reduction of computational complexity by searching only through the information sets that belong to the player using the algorithm. The actions of the opponent are substituted by operations on the information sets that could possibly be the effect of the opponent's actions. For example in visibility-based pursuit evasion domain [40], the actions of the pursuers are all possible moves of the agents and the actions of the evader are substituted by operation revealing the position of the evader on each position that became visible by the previous move of the pursuer or to an information set representing all possible locations of the evader that are not currently observable.

This algorithm is quite efficient, but it cannot be directly used the APRG. The main problem is that the algorithm cannot work with chance nodes that are crucial for our application. Moreover, the algorithm is based on the fact, that the actions of the opponent can directly influence the information set of the player. In our game, the actions of the opponent influence the information set only indirectly and probabilistically.

Paranoid Search

The goal of the paranoid search is to find a strategy that maximizes the utility of the defender against the worst-case attacker. We describe the action-selection in each choice node of the defender

and back-propagation of the utility values in each node (in each node we assume that we received the values from each of its successors).

The leaves of the game-tree propagate back their assigned value (either utility value of a terminal node, or an utility value estimated by an evaluation function in case the end of the look-ahead l). In chance nodes (nodes of the Nature player) value propagated to its predecessor is the expected utility of that node – each value returned from a successor $\sigma_n(h, r)$ is multiplied by associated probability π_r of that choice (i.e. either π_{pos}, π_{neg}, or π_a depending on the position of the node h in the game-tree).

In the defender's choice nodes the algorithm selects an action that the defender is supposed to play and again propagates some value to the predecessors. As we are dealing with imperfect-information game, the defender cannot distinguish between nodes in the same information set. Therefore we select the same action and propagate the same value in all nodes from the information set. Let assume we are in a node $h \in H$, $\rho(h) = d$ and there exist some j for which h belongs to the information set $h \in I_{d,j}$. Now, as we assume that we are playing against the worst-case opponent, we need to select such action that maximizes minimal defender's utility value resulting from the action:

$$\max_{o \in A_d} \min_{h' \in \sigma_d(I_{d,j}, o)} u_d(h')$$

Finally, in the attacker's choice nodes we simply propagate the value received from its successors upwards. This is sound, because (1) all successors of an attacker's choice node are in the same defender's information set, hence they all have the same calculated utility value; (2) the selection of opponents action was made implicitly in selecting the defender's action – if (as the defender) we cannot distinguish between the states that are determined by the actions of the opponent we assume the worst case.

3.5 Proposed Method Preliminaries

This section first introduces alternative, more compact representation of adversarial plan recognition game (APRG), which is later used as the bases of our novel solution method. Then, we briefly present the main principles of the established Monte-Carlo Tree Search (MCTS) method, which was used as bases of our novel algorithm. Finally, we analyze possible generalization of its core component – the selection function – that would be applicable in imperfect information games.

3.5.1 Reduced Game Representation

The definition of the game above is a standard extensive-form game with imperfect information and chance nodes and, in principle, it can be optimally solved by a standard algorithm for solving the games, such as the linear program for sequence-form representation of the game[51]. The size of the linear program is polynomial in the size of the game tree. However, the size of the game tree is exponential in the number of actions of the players and the length of the game. The practical games we focus on have tens of actions for each of the players and are played for tens of moves. For example, if each player has 10 applicable choices in each decision node and the game is played for 20 moves (corresponds to 60 plies) the size of the full game tree is 10^{60}. As a result, even traversing the whole game tree is computationally intractable.

On the other hand, the tree of the game has a very specific structure caused by the limited amount of information available to the players. The attacker never learns any information about the

defender's actions and the defender learns all the information only via the chance nodes that have a limited set of outcomes. As a result, the information sets of the players include a relatively large number of nodes in the tree. This fact allows us to represent the game in a much more compact way without any loss of information or the expressiveness of the computed strategies. The basis of the representation is the concept of a signal tree [13]. A signal tree represents the space of all possible developments of a game from a single player's perspective. In general, plies of player's actions and observations are alternating in the tree. After any observation, a child corresponding to each of the possible observations that follow the player's decision is added. The signal trees are closely related to information-set trees [38]. The only difference is that in a signal tree, multiple observations of the player can lead to the same information set (there are multiple nodes in this information set), while there is only a single node per information set in the information-set tree.

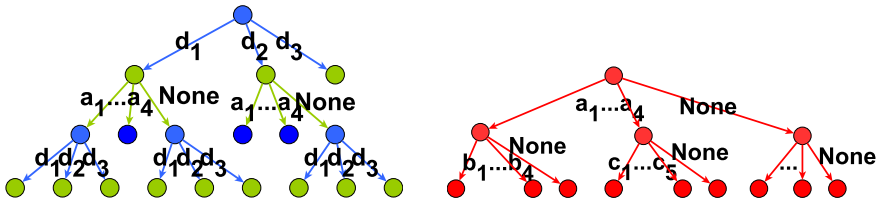


Figure 3.7: Signal trees of the defender and the attacker.

The attacker does not obtain any observation in the game. Hence, its signal tree is simply the attack tree with all meaningful attack plans (Figure 3.7). The nodes of its signal tree uniquely identify its information set. We denote the set of all attack tree nodes D_A.

The defender obtains an observation after each decision. Hence, the signal tree of the defender contains both observation and decision plys. We denote the observation nodes O_D and the decision nodes D_D. I general, each decision node has exactly m children and each observation node has exactly n children. However, forward pruning based on the domain-specific knowledge can be applied. The depth of the defender's signal tree is uniform and it is two times deeper than the length of the longest attack plan.

Any branch from one tree in combination with a branch of the same length from the other tree defines a unique history h in the extensive-form game – the full game tree. All histories that correspond to the same node in the signal tree of a player belong to the same information set of the player. Also, all information sets that exist in the full EFG are represented by a node in one of the signal trees. Moreover, the children of the node represent all applicable actions of the player in the information set, hence, any strategy of a player can be represented as probabilistic decisions on its signal tree.

The size of the two signal trees is much smaller than the size of the full game tree. With the 10 choices and 20 moves, the size of the attacker's signal tree is 10^{20} and the size of the defender's tree is 10^{40}. It is still not possible to traverse the signal tree in a reasonable time, but the reduction is substantial. In order to make this representation of the game equivalent to the full game tree, we have to solve two issues. First, the chance nodes in the defender's tree must become more complex. In the full game tree, the chance nodes selected the observation based on a fixed probability distribution. The distribution could have been different in each node. One chance node in the defenders signal tree corresponds to many chance nodes in the full game. For a fixed sequence of defender's choices, each possible sequence of the attacker's actions leads to a different chance node in the full game tree. But with the signal tree representation, all the attacker's sequences of the same length lead to the same chance node. If the chance node were a simple probability distribution, some of the information would be lost. On the other hand, the semantics of the chance nodes are the observations of the action currently performed by the attacker. For the fixed

choices of the defender, the probabilities of these observations depend only on the current (last) action of the attacker. Hence, if the chance node stores a conditional probability $P(A_n|A_a)$, we do not lose any information about the distributions.

The second issue are the utilities. In general, any leaf of the full game tree can have a different utility. The information about the utility cannot fit into the signal trees. That is why it has to be stored/computed separately. This should not be a problem since large games are generally defined in a succinct form, which includes an algorithm that can quickly compute the utility of any history in the full game. For any two branches of the signal tree, the corresponding history in the full game can be easily computed and consequently the same algorithm for computing utility can be used.

3.5.2 Monte-Carlo Tree Search

The Monte-Carlo tree search (MCTS) algorithm is recently a very popular algorithm for solving large perfect information games. It gained a lot of attention when it strongly improved the strength of the programs playing the game of GO and allowed creating programs comparable to human masters with minimal penalty. MCTS mains a part of the game tree close to the initial state. At the beginning, this tree consist of only the root node and then it gradually grows more in the areas that are promising for at least one of the involved players. The main step of any MCTS algorithm consists of four procedures that are repeatedly called in the following order:

- **Selection:** Starts in the root note and descends down the already constructed part of the tree based on the statistical information stored in the tree nodes, estimating their quality and confidence bounds.
- **Expansion:** After it reaches the leaf of the already constructed tree, it adds a small sub-tree rooted in the leaf to the maintained tree. Usually, not more than sub-tree of depth one is added.
- **Simulation:** It chooses one of the leafs of the newly added sub-tree and runs a simulation of the play following that point using random actions for each of the players and evaluate its result.
- **Backpropagation:** It returns back to the root of the tree and on the way up, it updates statistics in all the nodes using the result of the simulation.

The main idea of the algorithm is to use earlier iterations to create statistics that allow guiding the later iterations to the portions of the search space that are more relevant for the game. The output of the algorithm are the actions in the root node that lead to statistically most favorable results for the agents that runs the algorithm. One of the main advantages of this method in perfect information games is that it does not require strong heuristic evaluation function, as it is in classical minimax-like search. The simulation can be run until the end of the game, where the value of the game can be determined exactly.

3.5.3 Suitable Selection Function

The core component of MCTS is a selection strategy that optimizes the trade-of between exploration and exploitation. The exploration is using more the seemingly better actions to guide the

search to the more relevant parts of the tree. The exploitation is trying even the worse actions often enough to be confident that they actually are worse.

The problem of exploration versus exploitation is formally studied on the model of multi-armed bandits [24]. Hence, we refer to the method solving it as the bandit methods. The bandit method most commonly used in MCTS is UCB1. The combination of MCTS and UCB1 is called UCT [30]. It has been suggested for perfect-information alternating-moves games, in which it assures convergence to the same results as the minimax algorithm. From the game-theoretic viewpoint, the result of the minimax algorithm is optimal in these games. It is allowed by the fact that there is always a pure Nash equilibrium in the games.

However, vast majority of games of imperfect information or even the games with perfect information and simultaneous moves does not have a solution in the form of pure Nash equilibrium. In that case, UCT is not guaranteed to converge to the optimal solution of the game [50]. For this reason, we analyze alternative selection mechanisms for MCTS that would guarantee convergence to mixed Nash equilibrium in games with simultaneous moves and possibly in the more general imperfect information games.

Requirements and Related Work

A study, which is most relevant to this problem was performed by Teytaud et al.[55]. Instead of the UCB1 selection function, they work with the EXP3 function [4] that captures the optimal trade-off between exploration and exploitation for stochastic, or even adversarial, simultaneously acting agents. The authors in [55] compare the quality of game playing of MCTS with UCB1 and EXP3 and show superior performance of EXP3 on the game of Urban Rivals. However, this study uses a mixture of UCB1 and EXP3 nodes in the tree and does not give any details about parameter setting and numerically stable implementation, which appears to be crucial in application of EXP3.

EXP3 was successfully used also to play the game of tic-tac-toe with imperfect information in [5]. The approach is similar then the one we suggest for the adversarial plan recognition games. The algorithm is building a separate information-set tree for each of the players and performs the MCTS simulations in both trees simultaneously.

We have selected EXP3 as the selection mechanism for our approach as well. Here we analyze our requirements and the motivation for this choice. We first analyze the simpler case of MCTS for games with simultaneous moves and then generalize the observation to the general imperfect information setting. Applying MCTS in game with simultaneous moves is in principle very similar to playing a sequence of repeated games with high uncertainty about the pay-offs of the games. At the beginning, the algorithm has no information about the quality of individual actions and their combinations. The situation closely resembles the unknown game setting from multi-agent learning theory. There are several multi-agent learning algorithms that are able to learn optimal strategies in repeated play of an unknown game. Different classes of the algorithm have different requirements and guarantees.

We aim to practical applications, so we have to avoid many algorithms that focus on showing some specific convergence properties and do not provide efficint ways to converge quickly. For example, a method called Regret Testing [21] suggests a method that randomly chooses a strategy from a set of all possible strategy and tests the regret it causes. If it is too high, it randomly chooses a different strategy.

Fictitious Play is one of the first learning rules that lead to convergence to Nash equilibrium of a game. In each iteration, it uses computes the best response to the current strategy of the opponents. In order to compute the best response, it needs the statistics on opponent's past actions and the exact payoff matrix of the game. The first requirement could be satisfied in the simultaneous moves setting, but the second prohibits using fictitious play even in this context.

No-regret Learning A more promising source of possible selection strategies for MCTS is No-regret learning. It is a wide class of learning algorithms that allow selecting the best actions out of a fixed set without putting any assumptions on the process that generates the payoffs. More exactly, a learning algorithm exhibits no external regret if it can guarantee that the average regret per iteration converged to zero. If two players play a repeated zero-sum game using a no-regret strategy, their frequencies of using individual actions converge to the Nash equilibrium [17].

Even though no-regret learners converge to Nash equilibrium, some of the algorithms are still not usable for our purpose. For example, [17] presents a no-regret learning mechanism which allows convergence with average regret scaling as $O(\ln T/T)$ if both players honestly follow a protocol. This could be achieved in simultaneous moves games, but it would be much harder to achieve in general imperfect information setting, because the opponent can be in a different node of his tree in each repetition of the games. Moreover, this approach requires a common assumption of so-called Informed Online Decision Problem that cannot be satisfied even for simultaneous-moves games. It assumes the agent is informed also about the payoffs of the actions it did not choose in the current iteration.

If we drop this assumption and use only the reward of the applied action in the algorithm, we have Naïve Online Decision Problem. It fully respects our requirements. It assumes only the knowledge about the set of possible rewards (interval is sufficient), number of actions available to the player and the reward obtained for the selected action.

3.5.4 Exp3.1 Implementation Details

We base our implementation of EXP3 algorithm mainly on [3]. The main difference from the original paper [4] is that the algorithm maintains an estimate of aggregate pay-offs for each arm instead of the weights, which in [4] quickly grow to the limit of computer floating point number's precision. We use the following variables to describe the algorithm:

- K – the number of actions available to the agent.
- p_i – the probability of using action i in the current round.
- g_i – the reward gained for using action i in the current round normalized to interval $[0, 1]$. We define this value to be zero for the actions that are not used in the current round.
- $G_i = \sum_{s=1}^{t} \frac{g_{i,s}}{p_{i,s}}$ – is the cumulative gain.
- $\gamma \in (0, 1]$ – is the exploration parameter representing the probability of using a random action.
- f_i – the number of times action i has been used

The algorithm EXP3 follows.
$p_i = 1/K;\ G_i = 0;\ f_i = 0$

for $t = 1, 2, \ldots$ **do**
  Draw action a from distribution p.
  $f_a = f_a + 1$
  $G_a = G_a + \frac{g_a}{p_a}$
  $p_i = (1-\gamma)\frac{\exp(\frac{\gamma}{K}G_i)}{\sum_{k=1}^{K}\exp(\frac{\gamma}{K}G_k)} + \frac{\gamma}{K}$
end for

This algorithm works well in theory, but in practice, the exponential terms quickly grows to infinity. This can be partially solved by a sufficiently small exploration parameter γ. However, in MCTS, it is hard to use the parameter to keep the numbers within the range of the computer's precision. We need sufficiently large exploration to avoid getting stacked in local minima and we do not know how many iterations will we make in a node. That is why we transform the probability update rule to a computationally more stable form.

$$\frac{\exp(\frac{\gamma}{K}G_i)}{\sum_{k=1}^{K}\exp(\frac{\gamma}{K}G_k)} = \frac{\exp(\frac{\gamma}{K}G_i)}{\exp(\frac{\gamma}{K}G_i)(\sum_{k=1}^{K}\exp(\frac{\gamma}{K}(G_k - G_i))} = \frac{1}{1 + \sum_{k=1;\ k\neq i}^{K}\exp(\frac{\gamma}{K}(G_k - G_i)} \quad (3.6)$$

The second issue is the exploration parameter. If we want to assure convergence to the Nash equilibrium, the size of the parameter has to converge to zero. Otherwise, each action is chosen with probability at least γ/K which is not likely to be true in a Nash equilibrium. In order to guarantee the convergence to NE, [4] proposes algorithm EXP3.1. It runs the algorithm EXP3 in a sequence of epochs, each with a different setting of γ. The paper assumes restarting the algorithm EXP3 at the beginning of each epoch. This seems wasteful and we have better results with adjusting the parameter and continuing the algorithm. Hence our version of EXP3.1 is following.

$g_r = (K \ln K)/(e-1)$; $\gamma = 1$
loop
  Execute one iteration of EXP3 with current γ.
  if $\max_i G_i > g_r - K/\gamma$ **then**
    $g_r = g_r * 4$; $\gamma = \gamma/2$
  end if
end loop

The algorithm exactly follows the equations in [4]. If the estimate of the cumulative reward of the best action reaches the current bound g_r, the exploration parameter is decreased and the new bound is set.

3.6 Proposed Method for Solving APRG

The size of the game is too large to compute even an approximation of the optimal solution with a strict quality guarantee. Most methods that attempt to do that need to represent the strategy in each information set of the game. For the defender, each sequence of observations defines a valid information set. With 10 observations and the depth of the game around 20 moves, the size of the data structure is prohibitive. That is why we choose tree search techniques to play this game.

3.6.1 Concurrent Monte-Carlo Tree Search

The method we use for computing the strategy in the game is based on concurrent construction of both the attacker's and the defender's tree in Monte Carlo manner. It is a generalization of the Monte Carlo tree search [30] to imperfect information games. A similar approach has been proposed by [5] and applied to the game of phantom tic-tac-toe. However, we had to extend the algorithm in two directions to make it applicable to our problem. The first one is handling of the conditional chance nodes that appear in the defender's signal tree. The second extension is enabling the algorithm to improve the estimate of the solution of the game during the game play, i.e., designing an algorithm for progression in the game tree.

The algorithm maintains two MCTS-like trees corresponding to the two signal trees defined in Section 3.5.1. We present the pseudocode of the algorithm in Figure 3.8. At the beginning of the algorithm, both trees contain only one node – the root. We denode the root nodes *aRoot* and *dRoot* for the attacker and the defender respectively. The main loop of the algorithm performs identical iteration until it is out of time. It is an anytime algorithm. The more computation time it has allocated the better is the result it produces. In each iteration, first the plan for the attacker is selected (procedure *selectAttack*). Starting from the root node of the attacker's tree, the actions of the attacker's plan are selected based on a suitable selection function, which balances the exploration and exploitation (lines 1-6). When the selection reaches a leaf of the tree and it is not the end of the attacker's plan, new successors of the node are added to the tree based on the attacker's plan library. All the actions that can continue after the current history in the tree are added to the tree (line 10). Then, the plan is competed randomly to a full plan of the attacker (lines 13-17). At the end, the sequence of actions defining the attacker's plan is returned $(a^1 \dots a^h)$.

Next, a plan for the defender is selected in a similar way (procedure $selectDefence$). In the defender's decision nodes in the defender's tree, a suitable MCTS selection function is applied (line 4). Remember that in the observation nodes, conditional probability distributions $P(o_j|a_i)$ are stored. The algorithm has first selected the plan for the attacker, hence, we can identify the right distribution based on the action applied in the attacker's plan in the stage corresponding to the current node in the tree (line 7)[6]. If the observation node is already the leaf of the stored part of the table, the tree progression is broke (line 10). Otherwise, the child of the observation node is selected randomly according to the distribution (line 12). The selection of the defender's plan continues until either depth h (i.e., the length of the attacker's plan) is reached or a leaf in the defender's tree is selected. In the latter case, the nodes that are the children of the selected leaf are added to the tree (lines 16, line 23) and the state of the variables is modified so that the simulation part can begin. If the depth of the defender's plan is not still equal to the length of the attacker's plan, the rest of the defender's actions are selected randomly and the resulting observation are computed (lines 26-32). At the end of this stage, we have the full defender's plan and the observations it produced $(d^1, o^1, d^2, o^2, \dots, d^h, o^h)$.

With the plan of the attacker and the defender's observations, we can compute the utility function of the players in the $Main$ procedure (line 4). The utility is then distributed in both trees as in MCTS. Starting in the leaf of the trees where the samples left the stored trees, the statistics in the nodes are updated. Afterwards, next iteration of the algorithm starts.

[6]Here we assume that the observation depends only on the attacker's and the defender's action. It is so if the quality of the classifiers does not change during the game.

Procedure: Main

```
1: loop
2:    (aLeaf, aPlan) = selectAttack(aRoot)
3:    (dLeaf, dPlan, Observations) =
                        selectDefense(dRoot, aPlan)
4:    u = utility(aPlan, Observations)
5:    backPropagate(aLeaf, u)
6:    backPropagate(dLeaf, u)
7: end loop
```

Procedure: selectAttack(aRoot)

```
1: curNode = aRoot
2: while curNode is not leaf do
3:    action = select(curNode)
4:    aPlan = aPlan + action
5:    curNode = child(curNode, action)
6: end while
7: if curNode is plan end then
8:    return (curNode, aPlan)
9: end if
10: addNewChildren(curNode)
11: aLeaf = select(curNode)
12: curNode = aLeaf
13: while curNode is not plan end do
14:    action = random(curNode)
15:    aPlan = aPlan + action
16:    curNode =
              createNodeAfter(curNode, action)
17: end while
18: return (aLeaf, aPlan)
```

Procedure: backPropagate(node, u)

```
1: updateStatistics(node,u)
2: if node is not root then
3:    backPropagate(parent(node),u)
4: end if
```

Procedure: selectDefense(dRoot, aPlan)

```
1: curNode = dRoot
2: while curNode not leaf & length(aPlan) > 0
   do
3:    aAction = popFirst(aPlan)
4:    dAction = select(curNode)
5:    dPlan = dPlan + dAction
6:    obsNode = child(curNode, dAction)
7:    obs = confMatSample(dAction, aAction)
8:    Observations = Observations + obs
9:    if obsNode is leaf then
10:      break
11:   else
12:      curNode = child(obsNode, obs)
13:   end if
14: end while
15: if curNode is leaf & length(aPlan) > 0 then
16:   addNewChildren(curNode)
17:   dAction = select(curNode)
18:   dPlan = dPlan + dAction
19:   obsNode = child(curNode, dAction)
20:   obs = confMatSample(dAction, aAction)
21:   Observations = Observations + obs
22: else
23:   addNewChildren(obsNode)
24:   curNode = child(obsNode, obs)
25: end if
26: while length(aPlan) > 0 do
27:   aAction = popFirst(aPlan)
28:   dAction = random(curNode)
29:   dPlan = dPlan + dAction
30:   obs = confMatSample(dAction, aAction)
31:   Observations = Observations + obs
32: end while
33: if curNode is leaf then
34:   return (curNode, dPlan, Observations)
35: else
36:   return (obsNode, dPlan, Observations)
37: end if
```

Figure 3.8: The concurent monte carlo tree search algorithm for the first stage of the game. Procedures *select* and *updateStatistics* implement some MCTS selection strategy, such as UCT of EXP3.

Convergence of the Algorithm

We have implemented this algorithm under the hypothesis that it converges to the Nash equilibrium of the game. We did not have time to analyze this hypothesis in more details within this project so we do not provide a formal proof of this hypothesis. However, it will be a subject of our further research.

It has been proven that in a one stage normal form game, the overall frequencies of using individual actions with selection algorithm EXP3.1 converges to the Nash equilibrium, even if the exact utilities in the game and the number of actions of the opponent are unknown to the players [17]. This fact has been used to create an algorithm for playing an imperfect information version of tic-tac-toe [5], which is in spirit very similar to our algorithm. It uses separate trees for the players, but does not deal with chance nodes and game progression. The author claims that even in this setting, his algorithm converges to the NE of the extensive form game. However, he does not provide a formal proof or extensive experimental evidence to support this claim. Anyway, his as well as our experimental results indicate that this method can be used to create successful players for imperfect information extensive form games and we believe that it should be possible to proof the convergence for at least some restricted classes of games.

3.6.2 Continuous Reasoning

The algorithm described above finds a suitable course of action for both players at the beginning of the games. In theory, the iterative algorithm can be run for a long time at the beginning and the computed tree can then be used for playing the game without further computation. This approach is not applicable in practice for several reasons. First, the size of the tree that would need to be stored is huge. Second, the amount of computational time needed to compute a good strategy for a node increases exponentially with the depth of the node in the tree. Third, the information about the quality of the classifiers can change during the game. For these reasons, we choose to compute the actions gradually, adding more iterations after each step of the game. We call the operation on the pre-computed game trees at the beginning of the next stage of the game *progression in the game tree*.

Bayesian Tree Fixation

We further describe the algorithm for the defender, which is the main focus of this project. The progression in the game trees is performed separately for each of the trees. In the defender's tree, the progression is simple, because each player knows its own actions. Based on the selected action and the actual observation in the game, the root of the defender's tree is substituted by the corresponding node on the second level of the tree.

The situation is more complex in the attacker's tree. The defender does not observe the attacker's action so he cannot be sure about its current tree node in the later stages of the game. Instead, the defender maintains a probability distribution among the possible nodes where the attacker can be. Instead of moving the root of the tree, after each stage, one ply of the attacker's tree is fixed. Each node in the ply is substituted by a chance node with a fixed probability distribution computed by Bayesian inference. The probability distribution computed by the concurrent MCTS in the previous stage of the game serves as the a priory probability. The actual observation of the defender is used as the evidence, which influences the probability trough the confusion matrix corresponding to the classifier selected by the defender.

If the defender observed an observation o_j and the a priory (i.e, game theoretic) probability of the attacker performing a_i is $P(a_i)$ then the a posteriory probability can be computed as follows.

$$P(a_i|o_j) = \frac{P(o_j|a_i)P(a_i)}{P(o_j)} = \frac{P(o_j|a_i)P(a_i)}{\sum_{k=1}^{n} P(o_j|a_k)P(a_k)} \tag{3.7}$$

The probabilities of different observation given an attacker's action can be extracted from the confusion matrix that corresponds to the classifier selected by the defender in the game.

After an upper part of the tree is fixed, the following iteration in the attacker's tree proceed from the root node solely based on the fixed probability distributions and the back-propagation of the simulation value stops as soon as it reaches a fixed node. The ply following the previously fixed ply takes place of the new root of the game and in the next stage, all the nodes in this ply is fixed in the same way as described above.

Example 1. *Assume there are three actions applicable in the root of the attacker's tree in the first stage of the game. After we run the concurrent MCTS for a period of time, the frequencies of using the three actions converge to*

$$P(a_1) = 0.67\ ; P(a_2) = 0.33\ ; P(a_3) = 0.$$

Further assume that the defender used action d_2 with the confusion matrix as in Figure 3.5 and it obtained the observation o_2. The a priory probability of the observation based on the confusion matrix is

$$P(o_2) = P(o_2|a_1)P(a_1) + P(o_2|a_2)P(a_2) + P(o_2|a_3)P(a_3) = 0.163.$$

Then the probabilities of attacker's actions we fix in the root node based on equation (3.7) are

$$P(a_1|o_2) = 0.4\ ; P(a_2|o_2) = 0.6\ ; P(a_3|o_2) = 0.$$

Low Probability Pruning

With the Bayesian fixing of the attacker's tree, the attacker builds a probability distribution over all possible plans of the attacker based on the approximation of its rational behavior and the noisy observations produced by the classifiers. Unlike the tree of the defender, which is strongly reduced by each progression, the attacker's tree is only growing during whole game. With each stage of the game, the number of nodes in the attacker's tree that may be the attacker's current node increases exponentially. On the other hand, most of them will have very little probability. In order to keep the computational complexity of the game progression low and to free some of the memory allocated by the attacker's tree, we decided to prune the branches of the attacker's tree with probability lower then certain threshold. We chose the threshold to be 1%.

Simple pruning The first pruning algorithm is less radical and we use it in the proof of concept implementation of the algorithm. It prunes some parts of the tree, but it should not influence the computed results substantially. It first fixes the probabilities in a node as described above. Then it removes the actions with probability smaller than the threshold and their respective subtrees from the tree. This algorithm reduces the size of the tree, but still allows exponential growth of the number of the candidates for the current node in the attacker's tree.

Strong pruning The second pruning algorithm we suggest uses a limited number of possible current node candidates in each stage of the game. The algorithm stores a subset of nodes from

the current ply in the attacker's tree that are most likely to be the current attacker's node. With each node, it stores the probability that this node is actually the current node of the attacker. In the first stage, this set contains only one node – the root – with 100% probability. After an observation is received, the algorithm computes the probability for all children of the nodes in the current subset. For each node, if the probability of some of its children is lower than the threshold, the children are removed. If any of the children are left, they are renormalized to sum to the probability of their parent and inserted to the subset of nodes for the following stage. With this pruning and the threshold of 1%, there are never more than 100 root candidates in the current set.

3.7 Adversarial Plan Recognition Game in CAMNEP

There are several specific problems we need to address if we want to apply the algorithm from the previous section to the network security domain in combination with CAMNEP.

3.7.1 Problem Mapping

We have designed the adversarial plan recognition game to be applicable to various domains, where one player tries to execute a plan undetected and the other player uses noisy sensors to observe the plan. However all the time, we had in mind also the application to network security, which is the goal of this project. That is why the problem maps naturally to the defined model of APRG. In the implemented system, one stage of the game corresponds to the 5 minutes long interval used for anomaly detection in CAMNEP. The attacker has 13 basic actions that can be composed to plans of various lengths based on their preconditions and the information gained by their execution (see Section 2.3). Based on our assumptions, we assume that each 5 minutes long interval contains exactly only one action of the attacker. The defender is represented by an intrusion detection system. It has 22 different settings (corresponding to aggregation operators) that can be selected as defender's actions. The confusion matrices defining the quality of the operators are estimating using challenges (see Section 2.4) and further discussed in the following section.

The presented model of APRG does not consider the false positives explicitly. They are a very important factor in the network security domain, because the attention of the network administrator is a very scarce resource. The importance of the false positives can be reflected in the confusion matrices. All the probability values corresponding to no action of the adversary and some observation of an action by the classifier are the false positives. Similarly, the values classified as no action even though some action was performed are the false negatives. Most classifiers allow trading one of these values for the other. If the classifier is not certain that it can classify an action correctly, it can still report that no action has been seen to reduce the number of false positives. This will naturally be reflected in the quality of the classification, but it is already represented in the model in the form of confusion matrices. In our proof-of-concept implementation, we did not focus on this aspect and we used the classifiers in the form as they were suggested by CAMNEP. Moreover, in our implementation, the confusion matrices are computed only form the challenges modeling malicious traffic, so false positive situations do not occur in the confusion matrices at all.

3.7.2 Validity of the assumptions

From the perspective of the application domain, we can divide the assumptions needed by the model to two categories. The first category are assumptions that either hold in the model or can be shown not to have a major impact on the performance of the solution. The second category are temporary

assumptions, which does not fully hold in the real world domain, but importantly simplify either design or computational complexity of the model. The later assumptions are very important in research methodology, because they allow solving a simpler problem and then gradually add complexity by removing the extra assumptions. If the system does not perform well even with the additional assumptions, it is not likely to work without them and the core ideas should be redesigned. Otherwise, the main principles seem promising and the research should focus on whether and how it is possible to remove the additional assumptions.

Justifiable Assumptions

First we review the assumptions of the model that are reasonable in the network security domain.

Both players are rational This adversarial planning recognition game is a part of complex intrusion detection system that focuses on detecting sophisticated attacks. A sophisticated attacker knows exactly what he is after and has resources and experience to design very good strategies how to achieve it. If the defender uses the developed method, it also approximates the rational behavior. Hence, this assumption is reasonable.

Both players know the full plan library of the attacker This assumption usually does not fully hold in real world computer networks, but we expect that it does not prevent the proposed system from performing well. Large portion of the attack tree is certainly known to both players. This project targets highly sophisticated attackers that can be assumed to have extensive knowledge of their options. Moreover, the actions that model the behavior of the attacker in our system are quite abstract and high amount of specific attack plans can be mapped to one of the plans in the attack tree. Hence, if this assumption does not hold, we expect that the either the attacker or the defender knows about a few attack plans that are not known to the opponent. If the defender computes with a plan that is not known to the attacker, it can be slightly over-cautious. The defender can be able to gain a slightly more as in the case that the defender does not use the plan in its computation. On the other hand, it still guarantees the computed worst case quality of its performance. The situation is worse if the attacker knows about an attack plan not considered by the defender in its computation. In that case, the guarantees provided for the defender do not hold anymore. However, it is very likely that the executed plan will be similar to some of the considered plans and the quality of the classification will be reasonably good. Also, this situation corresponds to the zero-day attacks. The original non-strategic CAMNEP system is designed to perform well on these kinds of attacks. We expect the original and the strategic systems to run one next to each other. Even if the novel plan avoids the game theoretic detection system, it can still be detected by the other one.

The available classifiers and their quality are known to both players This assumption trivially holds for the defender, as the defender knows the classifier it uses and it can estimate their quality using challenges. This assumption is reasonable also for the attacker. CAMNEP, as most of the intrusion detection systems, can be obtained and reverse-engineered by the attacker. Most of the principles behind the intrusion detection systems have been published and as such are available to the attacker. In order to get a good idea about the current quality of the classifiers, the attacker just needs to model the characteristics of the network traffic in the target network. This can be done exactly if some part of the network has been already compromised, or approximated with sufficient knowledge about the structure of the network. Again, these assumptions represent the worst case in some aspect.

The defender can use only one classifier at a time This assumption does not restrict the applicability of the model too much. As in the case on CAMNEP, even if the IDS can use multiple classifiers, their reports are somehow aggregated at the end. After the aggregation, the classifiers already behave as a single classifier, which consumes all the defender's resources. Having more insight into the internal structure of the classifier and recommending combinations of sub-classifiers could improve the performance of the system in the future, but we chose not to do that in this project.

The quality of the classifiers does not change during the game This assumption clearly does not work in CAMNEP and we discuss it in Section 3.7.3.

Simplifying Assumptions

We continue with the assumptions we adopted to simplify the problem. These assumptions must be removed in order successfully apply the developed system in a generic real world computer network.

Single attacker who executes single plan In a real world computer network, multiple attackers will attack the network simultaneously and the attacks and the information gained in their progress can be shared and coordinated among the attackers. To model such highly complex concept of the attacks, a more sophisticated algorithm will need to be employed. For example, in the case that we can assign the actions to individual attackers, we can run multiple copies of the algorithm, each of which will play against one of the attackers and aggregate the results (e.g., play against the most dangerous attack). The applicability of this or similar approaches will have to be addressed in future research.

All actions of the attacker have equal length In reality, some actions of the attacker, such as connecting to an infected host, take much shorter time than the other (e.g., horizontal ping scan). As a result, more actions of the attacker can be executed in single stage of the game or an action can take several stages. We believe that in case the underlying IDS can provide a sequence of observations for single stage of the game, the proposed approach can be adjusted to work without this assumption. However, allowing a sequence of observations in one stage would most likely mean further increase in computational complexity of the approach. I.e., more time would be needed to compute good solutions, or worse solution would be achieved in the limited time.

Noop actions not used From the perspective model expressiveness, this assumption could be removed immediately. Noop can be modeled simply as any other action. From the computational complexity perspective, removing it strongly simplifies the problem. The main reason is that the attacker could insert arbitrary number of noop actions into its plan, making the space of the possible plans much larger. This issue could be partially tackled by using discounted utility to limit the length of the attacker's plans, but it will most likely require a more sophisticated solution to make to approach applicable it real world domains.

3.7.3 Changing Confusion Matrices

The general algorithm in for solving APRG described above assumes that the quality of the classifiers represented by the confusion matrices does not change during the game. This is not true in CAMNEP. The quality of different operators used by CAMNEP strongly depends on the current and the recent state of the network. The classifiers are continuously adapting and their quality for each stage is estimated using the challenges as described in Section 2.4.

If we had a model describing how the confusion matrices change over time, we could modify the algorithm to use some prediction of the confusion matrices in future stages of the game. However, the network traffic is very unpredictable and any model estimating the quality of classification in advance would be highly imprecise. The most exact estimation of the future classifier's quality seems to be their current confusion matrix.

Without the ability to predict the quality of the classifiers, we use the same confusion matrices in all stages in the initial computation. As it is the best estimate of the future development and none of the players can do much better, we can expect rational strategies to be produced by the algorithm. However, if the confusion matrices change substantially between the stages, the soundness of the progression in the game tree is weakened. The players already know the new state of the network and all the iterations performed in the previous stages can cause irrational bias in the strategies. On the other hand, if the optimal strategy did not change a lot, the iterations form the previous stage can still guide the search. With some king of aging of the iterations, it can be a useful heuristic to improve performance of the algorithm. Nevertheless, we chose the simplest solution for our proof of concept implementation.

At the beginning of the new stage, we always drop the whole tree of the defender and we start next iterations only with the root node. The new iterations are performed with the updated confusion matrices. In the attacker's tree, we keep also the parts of the trees computed in the previous stages, because we assume that, especially in the later stages of the game, the base utility of the plans is very important. Hence, the optimal strategy of the attacker is less likely to change substantially.

3.7.4 Confusion Matrix Filtering

Each defender action has a confusion matrix associated with it. Confusion matrix gives a probability distribution over possibly observable actions given the actual attacker action taking place in the network. There are 22 defender actions altogether. This would introduce a high branching level to the game tree and require a much larger amount of samples of the node to calculate a good estimate of the quality of each of the actions. After direct inspection of the matrices, we have found out, that many of them provide a very similar, even near identical distributions for all of the attacker actions. As a result, we have decided to perform pruning of the defender actions based on the similarity of their respective confusion matrices.

There are multiple methods of performing this task. A basic method would be a comparison of two confusion matrices using each of the probabilities for each of the attackers and testing, whether the difference between the probability from the first matrix and the second matrix differ only by a small, given epsilon value. If the difference is greater, than the epsilon value for any of the probability pairs, then the whole matrices are considered to be different. One of the more established methods of probability distribution comparison is the Kullback-Leibler divergence.Kullback-Leibler (KL) divergence (also termed *information divergence* or *information gain*) is a standard measure used in probability theory to compute the similarity of two probability distributions. For two discrete probability distributions P and Q over the same random variable (the same feature space in our

Input: $CMs = (M_1 \dots M_n)$ – sequence of confusion matrices; ϵ - threshold
Output: $(M_1 \dots M_m)$ – a reduced sequence of confusion matrices

```
1: for M_1 ∈ CMs do
2:   for M_2 ∈ Out do
3:     similar = true
4:     for each attacker's action a do
5:       if D_kl(M_1[a]||M_2[a]) > ε or D_kl(M_2[a]||M_1[a]) > ε then
6:         similar = false
7:         break
8:       end if
9:     end for
10:    if similar=true then
11:      Out = Out + M_1
12:    end if
13:  end for
14: end for
15: return Out
```

Figure 3.9: The algorithm for reducing the number of actions used by the defender.

case), the KL divergence of Q from P is defined as

$$D_{kl}(P\|Q) = \sum_x P(x) log_2 \frac{P(x)}{Q(x)} \tag{3.8}$$

where x in the sum iterates over the range of the random variable. KL divergence in this form is not a metric per se, because it is not symmetric, i.e., $D_{kl}(P\|Q) \neq D_{kl}(Q\|P)$ in most cases. That is why the symmetrized form of Kullback-Leibler divergence is often used

$$D_{skl} = D_{kl}(P\|Q) + D_{kl}(Q\|P) \tag{3.9}$$

We use a similar metric. If both $D_{kl}(P\|Q)$ and $D_{kl}(Q\|P)$ are lower than a specified threshold, we consider the two distributions to be identical. Simple KL divergence is still not enough, because we are comparing a set of probability distributions, not just two distributions. We consider two sets of probability distributions identical, if the KL divergence value of each of the probability distributions, is smaller than the selected epsilon (1.2997615 – a hand-picked value) in both directions.

The algorithm for reducing the number of defender's actions is presented in Figure 3.9. It iterates over all the confusion matrices and tests each column of the confusion matrix for similarity based on the KL divergence. If a new matrix is similar to any of the previously analyzed matrices, it is discarded. If not, it is added to the output sequence.

The number of selected actions may vary from 1 to 22, and 1 has been observed to happen on a number of occasions. The game theoretic method is meaningful even with a single action of the defender, because the attacker's tree has to be modified according to the observation and explored with respect to this modification.

3.8 Experimental Evaluation

The experimental evaluation consists of three independent experiments. First, we evaluate the selection function used in concurrent MCTS on small matric games, then we use an entirely synthetic, small setup, removed from the network security domain, which can be easily analyzed by hand. At the end, we analyze a setup inside the network security domain.

3.8.1 Bandit Methods Evaluation

In order to evaluate the benefits of using EXP3 and assess the speed of its convergence in MCTS, we performed a set of experiments on the repeated unknown games setting. We focus only on the zero-sum games, because we have formulated the APRG as a zero-sum game.

We present the main evaluation in Figure 3.10. It compares the speed of convergence for different setting of the exploration parameter γ in five different zero-sum games. Both players were using the same variant of EXP3 and the exploitability of the strategy of the row player is presented.

The experiments show that with a very small exploration parameter, EXP3 converges slowly. On the other hand, larger exploration causes the algorithm to converge to a more uniform distribution, which is exploitable, especially in case a pure Nash equilibrium exists (see Figure 3.10h). Otherwise, the convergence is quite robust to the choice of the parameter. In the long run, the theoretically sound EXP3 performs best. We have evaluated also an alternative dynamic setting of the parameter and denote it autoNaN in the results. The parameter was decreased smoothly according to the following function.

$$\gamma = 0.2 * \frac{1}{2^{t/1000}} \tag{3.10}$$

Using this parameter setting, EXP3 tend to converge faster at the beginning, but the difference is not very large and it behaves worse with higher number of iterations. Based on these results, we further use the presented variant of EXP3.1 for setting the parameter.

The worst results for all settings of γ are achieved for the game in Figure 3.10d. EXP3 algorithm gets deceived by several high rewards on the second action while both players have more or less uniform strategies. Then, it requires over a million iterations to converge to a strategy with regret lower than 0.5. Eventually, even for this game EXP3.1 converges to low regret.

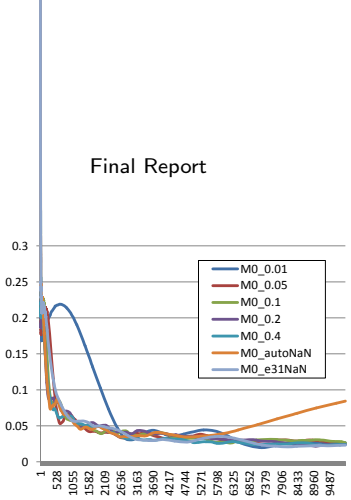
The following experiment compares the most successful EXP3.1 algorithm to the popular UCB1. UCB1 assumes that the samples of payoffs of individual actions origin from an unknown probability distribution. This is not true in simultaneous-moves games. As a results, UCB1 does not necessarily converge to NE in the repeated play of an unknown game [50]. One of the main reasons if synchronization of the deterministic exploration, which causes the rewards of the player to be strongly dependent on each other actions. The situation is even worse if the exploration parameters are identical. In this experiment, we use the parameter $C = 2$ for the row player and $C = 3$ for the column player. Despite of the suboptimal behavior, UCT has been successfully used games with simultaneous moves in the general game playing competition[22], hence, we evaluate it as well. The results are presented in Figure 3.11.

We can see that UCB1 performs reasonably well if only a small number of iterations is allowed. With more iterations, it gets worse for some settings. UCB1 seems to be working well when the optimal distribution is close to a pure strategy or a uniform strategy on some limited support. Generally, the experiments show that UCB1 may be a usable option for our MCTS implementation, but the theoretical guarantees for EXP3.1 result to a more stable performance.

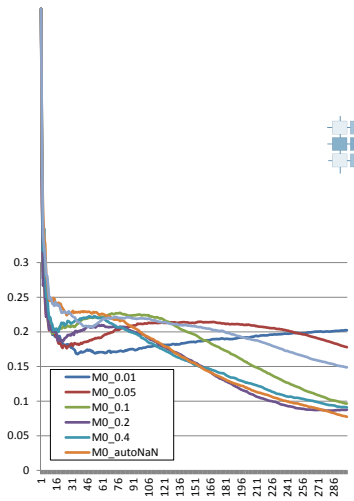
2	0
0	1

NE=$(\frac{1}{3}, \frac{2}{3})$

(a)



(b)

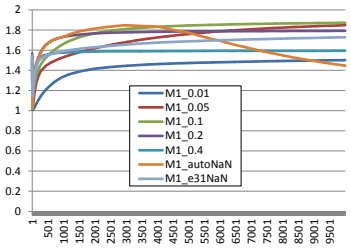


(c)

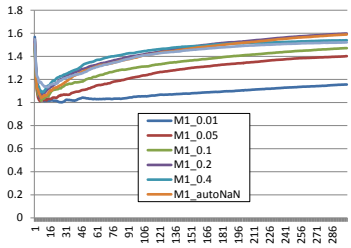
3	2
1	1000

NE=$(\frac{999}{1000}, \frac{1}{1000})$

(d)



(e)

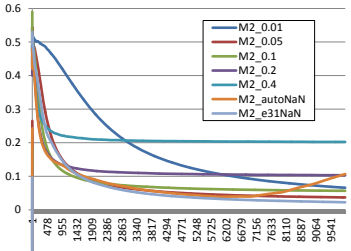


(f)

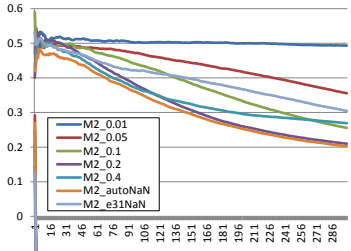
3	0
2	1

NE=$(\frac{999}{1000}, \frac{1}{1000})$

(g)



(h)

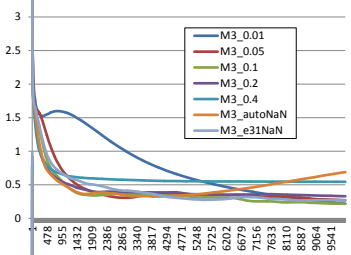


(i)

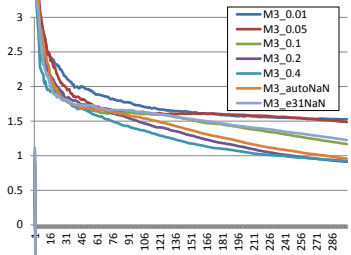
10	-5	0
-1	0	5
0	1	-10

NE=$(\frac{1}{16}, \frac{5}{8}, \frac{5}{16})$

(j)



(k)

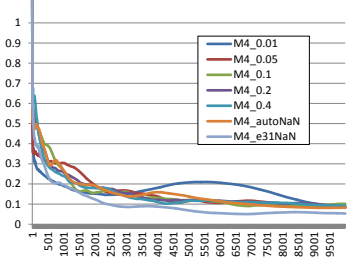


(l)

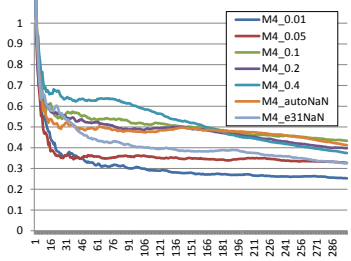
3	-1	2
1	2	-2

NE=$(\frac{4}{7}, \frac{3}{7})$

(m)



(n)



(o)

Figure 3.10: Dependence of the exploitability of the strategy given by relative frequency of use of actions in the EXP3 algorithm on the number of iterations. One row of the figures is always the game and its Nash equilibrium strategy for the row player, then follows the exploitability if the computed strategy over 10000 iterations and a detail to the first 300 iterations in the same graph.

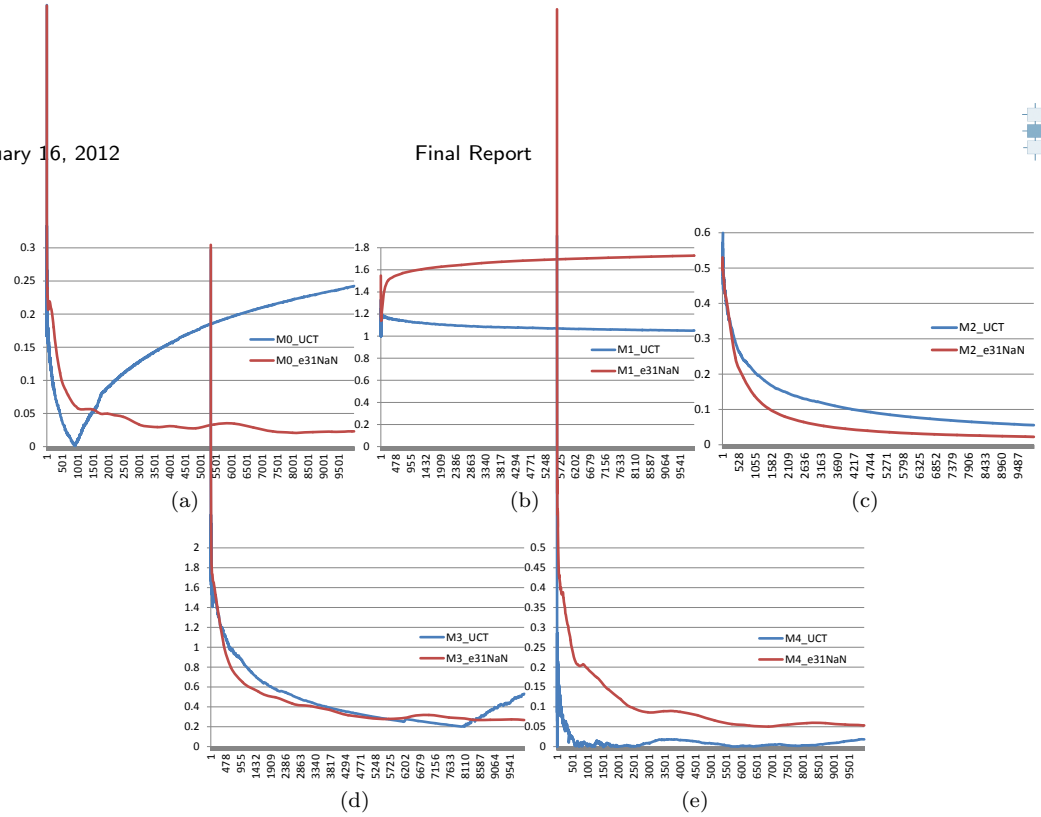


Figure 3.11: Comparison of exploitability of EXP3.1 and UCB1. The results (a)-(e) correspond to the matrix games in Figure 3.10 in the same order. The figures show the dependence of exploitability of the strategy given by the relative frequencies of using individual actions on the number of iterations.

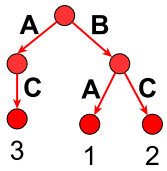


Figure 3.12: The attack tree for the simple synthetic experiments. The numbers in the leafs denote the utility for the attacker in case it has not been observed at all.

3.8.2 Synthetic experiments with APRG

There are two synthetic examples. Each of them provides the defender with two actions and the attacker with three actions. The attacker's actions are identical for both cases and based on their preconditions and effects, they create the attack tree depicted in Figure 3.12. The defender's domain in the first case is different from the one in the second case. The Nash equilibrium can be easily found in both synthetic cases. In the first case, we have a single pure Nash equilibrium, the second case contains a single mixed Nash equilibrium. Because of the small size of the domain, we have chosen to limit the evaluation time to 20 seconds for each of the two stages, i.e., 40 seconds in total. Almost one million iterations of the concurrent MCTS has been performed in each stage.

Case 1

The confusion matrices for the first case are presented in Figure 3.13. The first classifier ($d1$) can perfectly detect action A and B, but never detects action C correctly. If the attacker executes C,

*d*1

	NOOP	A	B	C
NOOP	1	0	0	1/3
A	0	1	0	1/3
B	0	0	1	1/3
C	0	0	0	0

*d*2

	NOOP	A	B	C
NOOP	1	1/3	1/3	0
A	0	0	1/3	0
B	0	1/3	0	0
C	0	1/3	1/3	1

Figure 3.13: Confusion Matrix for actions $d1$ and $d2$ in the first synthetic case

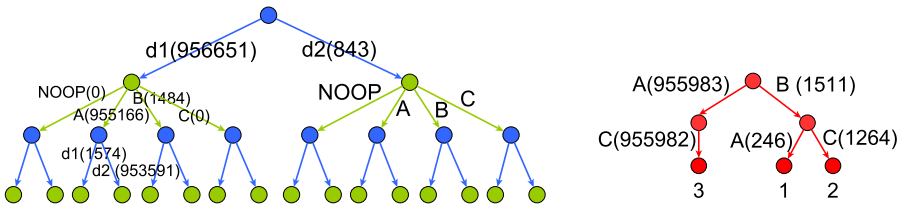


Figure 3.14: The empirical frequencies of using individual actions in the tree. Some of the values are omitted for clarity.

it can be detected as any other action with equal probability, but newer as C. Similarly, classifier $d2$ can flawlessly detect action C and never actions A and B. It is easy to see that in this case, the defender should always use $d1$ in the first stage of the game. Regardless of what happens in the first stage, the attacker will prefers to play C in the second stage. Hence, the defender prefers playing $d2$ in the second stage. It means that the pure Nash equilibrium for the first synthetic case is $[(A, C); (d1, d2)]$ and the value of the game is $2/3$.

The results of running concurrent MCTS with EXP3.1 in the first synthetic case are presented in Figure 3.14. The numbers at the edges are the number of iterations in which the individual edge has been selected. We can see that as in the case of the repeated matrix games, the empirical frequencies of using individual actions our algorithm also converges to the single Nash equilibrium. In both roots of the trees, the action which is not in NE has less than 0.2% of iterations. If we look at the second defender's decision after the most probable observation, the right action is also selected in more than 99.8% iterations.

Case 2

The second case evaluates the behavior of the algorithm in a game without any pure NE. The confusion matrices for the first case are presented in Figure 3.15. The first classifier ($d1$) can perfectly detect only action A and never detects the other two actions correctly. Classifier $d2$ detects only action B.

*d*1

	NOOP	A	B	C
NOOP	1	0	1/3	1/3
A	0	1	1/3	1/3
B	0	0	0	1/3
C	0	0	1/3	0

*d*2

	NOOP	A	B	C
NOOP	1	1/3	0	1/3
A	0	0	0	1/3
B	0	1/3	1	1/3
C	0	1/3	0	0

Figure 3.15: Confusion Matrix for actions $d1$ and $d2$ in the second synthetic case.

	A	B	$NE(probability)$
$d1$	3/2	2	0.8
$d2$	3	1	0.2
NE	0.4	0.6	

Figure 3.16: Matrix game equivalent the first stage of the second synthetic case under the assumption that the attacker acts rationally in the second stage.

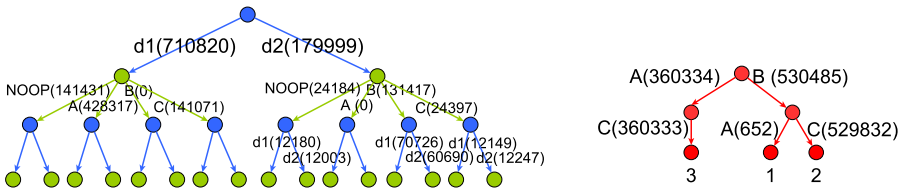


Figure 3.17: The empirical frequencies of using individual actions in the tree. Some of the values are omitted for clarity.

Regardless of what happens in the first stage, the attacker prefers to execute action C in the second stage, because it knows that the defender has no classifier that could detect this action and the action leads to the higher base utility. The important part of the game is player in the first stage. The game can actually be transformed to the matrix game in Figure 3.16. For example, if the attacker selects plan (A, C) and the defender selects the action $d1$, it will observe the action A and hence the utility will be 3/2. If the defender chooses the wrong classifier $d2$, it will not observe any of the actions in the attack plan and the utility of the action is 2. This matrix game does not have a pure NE and the only mixed NE is the one depicted in the figure. The attacker should choose B with probability 0.6 and the defender should choose $d1$ with probability 0.8.

The results of running the proposed algorithm are presented in Figure 3.17. The solution of our algorithm suggests that the attacker should use action B with probability 0.596 and that the defender should play action $d1$ with probability 0.798. This result is very close to the actual NE of the game.

Even though we did not manage to formally proof that the proposed algorithm always converges to NE, our synthetic experiments indicate that it may be the case.

Comparison to baselines

In the last experiment with the synthetic domain, we measured how the complete system including the progression in the tree responds to the tree plans of length 2 of the attacker. We run the computation for 20s per stage and for each plan, the whole system run 29 times. We present the mean of all these runs. The results are presented in Figure 3.18. Note that the plot depicts the attacker's utility and the defender's loss. Hence, the lower values are better for the attacker. We use the same baselines also for the experiment with real network data, so we explain them here in more details. The first value we measure is the best response (BR). It corresponds to selecting the best possible classifier in each stage of the game in case the defender knows the plan of the attacker in advance. This value is generally not achievable by a real system, which does not know the plan of the attacker in advance, but it creates a good lower bound on achievable performance. A similar value is the worst response (WR). It is the utility achievable in case of knowing the attacker plan in advance and intentionally selecting the classifier with smallest chance to discover the next attacker's action. In the case of our synthetic domain, the defender can always

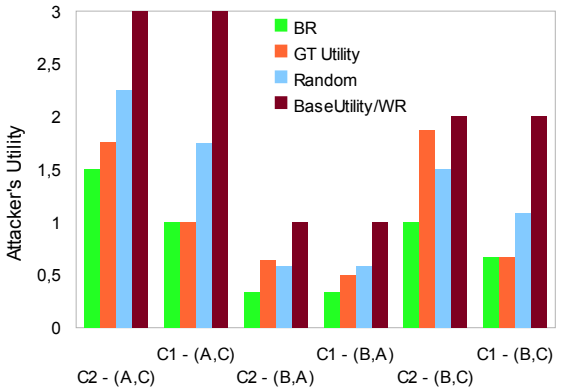


Figure 3.18: Comparison of the results of the proposed technique in comparison to baseline algorithms. C1/C2 means Case 1 and Case 2 of the synthetic experiment. BR is the value of selecting the best response classifier, Random is the value of selecting random classifier, BaseUtility is the value of the attacker's plan without any observation and GT is the value obtained by our method.

find a classifier that will certainly not see the next action; hence the value of the worst response is always the same as the attacker's base utility. The random classification just chooses any of the classifiers with uniform probability distribution. We refer to the results of the proposed game theoretic method as GT.

In the first case, the proposed game theoretic system perform better than the random classifier selection regardless of which plan has been used by the attacker. In case of the rational attacker's plan (A, C), our system has identified and used the best response classifiers. The same is true also for plan (B, C). The best response classifiers for action A is the same as for action B. Even though our system assumes the opponent playing B very improbable, because of ideal confusion matrix, it believes the opponent has played B after the observation and selects $d2$. This completes the best response. Only in case of the attacker playing (B, A), our system does not choose the (ex post) optimal strategy. It is because playing (already irrational) action B, the rational player would choose C. It is expected by the system and the classifier suitable for this action is played by the defender. However, the attacker plays A and the defender could be better off is he chose the other classifier. On the other hand, it should not know it in advance. Moreover, the utility obtained by the attacker this way is still lower than the utility, it would have guaranteed if it has chosen the other action, as we can see in the bars of plan (B, C).

In the second case, the rational attacker can play any of the plans (A, C) and (B, C). The later should be more probable. Only for the first plan, our approach reaches higher utility then the random selection of the classifier. There is a good reason for that. Note that the utility achieved for plans (A, C) and (B, C) are very similar. In fact, the difference is caused only by a small number of samples in the experiment. The defender does not know in advance if the attacker plays A or B, hence it is worth playing worse than random in case of (B, C) in order to prevent higher loss in case (A, C). The last attacker's plan – (B, A) is clearly irrational. The attacker gains much less then it would have guaranteed by other plans. Still, the game theoretic method plays close to the random strategy.

This experiment shows that in case of facing the optimal plans by the adversary, our game theoretic method provides good results. Moreover, if the opponent does not act rationally, it still can have reasonably good performance.

3.8.3 Full scale experiments with APRG in CAMNEP

The best evaluation of the proposed system would be to run CAMNEP with the game theoretic module on a labeled real world computer traffic that contains several samples of rational attack plans, which satisfy our assumptions. However, we have only one labeled data sample, which contains such a plan. We refer to the experiments on this sample as the experiments with the real plan. In order to make the experiments more representative, we needed more sample plans. That is why we used the real network traffic and injected artificial attacks generated by the challenge agents into the traffic (see Section 2.1).

Further, instead of running the whole CAMNEP with all the data many times to achieve statistical significance of the results, we have de-coupled the evaluation. The combined evaluation would take prohibitively long time. We first run CAMNEP on the data for several different random seeds. The random seeds influence mainly the challenge insertion process; hence they lead to different learning of the basic CAMNEP classification agents, which influences the quality of classification. For each seed, we logged the confusion matrices of all operators (i.e., classifiers) available in CAMNEP in each stage of the game. Using the confusion matrices as the input data, we run the evaluation of the game theoretic method separately.

We have evaluated our algorithm's performance by performing multiple repetitions over 10 rational attacker plans (see Figure 3.19) with an emphasis on the real plan, which reflects one of the most frequent attack plans. The real plan has been evaluated 32 times for each of its 3 seeds, while the other plans were evaluated only 20 times for each of their respective seeds. Seed refers to value that has been used to calculate confusion matrices and reflects the network traffic. Each of the seeds is specified by a sequence of confusion matrices for each of the defender's actions for each of the stages (of the attacker's plan).

The network domain evaluation data was created by artificially inserting the attacks in the real network traffic data. The source of the data for the real attack plan is the real network traffic, in which we have conducted this attack.

The provided traffic data spans a length of stages, during which the attacker's actions take place. In order to further simplify the already difficult task for the game theoretic model, we have selected only the stages during, which the actions of the attack occur. So there are no NOOP actions and actions do not span multiple stages. Only the appropriate confusion matrices are then taken into account in the evaluation.

For the algorithm pseudocode, which the evaluation follows see the Algorithm 3.20. The execution is simple. For each of the stages of the attacker's plan, the next action is requested from the defender. Using this action and the attacker's action from the given stage, the observation is generated from the probability distribution $P(o_i|a_j, d_k)$, which is specified by the appropriate confusion matrix. Then the game advances to the next stage by providing the observation, along with the next set of confusion matrices, to the defender and requesting the next action to take. The algorithm ends when the whole attacker's plan has been processed and all observations have been generated.

We are going to assess the quality of our game theoretic approach by taking averages at multiple levels. Here, we are going to provide the average utility achieved for each of the attacker's plans and an average utility over all plans. We are going to base this data on repeated runs of the approach for the same attack plan with the same traffic profile. We have also calculated the quality of the best response (BR), worst response (WR), the random response (Random) and each of the possible defender's action's (Actions), if used in each of the stages of the attack. The BR and WR was calculated with a prior knowledge of the attacker's plan by deliberately selecting

```
HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE, BRUTEFORCE, CONNECT_TO_HOST //real

DNS_REQUESTS, HORIZONTAL_PING_SCAN, PORT_SCAN, FINGERPRINTING, WEB_ATTACKS,
CONNECT_TO_C2 //malware_web

DNS_REQUESTS, HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE, HORIZONTAL_PING_SCAN,
SEND_SPAM, PORT_SCAN, DDOS_TO_SPECIFIC_SERVICE //ddos

HORIZONTAL_PING_SCAN, HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE, BRUTEFORCE,
FINGERPRINTING, CONNECT_TO_C2, DDOS_TO_HOST //malware_ddos

HORIZONTAL_PING_SCAN, HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE, PORT_SCAN,
DDOS_TO_HOST, DNS_REQUESTS, DDOS_TO_SPECIFIC_SERVICE, BRUTEFORCE
//ddos_and_bruteforce

HORIZONTAL_PING_SCAN, PORT_SCAN, HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE, BRUTEFORCE,
CONNECT_TO_HOST, DDOS_TO_SPECIFIC_SERVICE //bruteforce_and_malware_ddos

HORIZONTAL_PING_SCAN, PORT_SCAN, WEB_ATTACKS, DDOS_TO_HOST, CONNECT_TO_C2
//web_attacks_and_ddos

HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE, FINGERPRINTING, BRUTEFORCE, DNS_REQUESTS,
CONNECT_TO_HOST, SEND_SPAM //bruteforce_and_send_spam

HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE, HORIZONTAL_PING_SCAN,
DDOS_TO_SPECIFIC_SERVICE, WEB_ATTACKS, DNS_REQUESTS //ddos_and_web_attacks

HORIZONTAL_SCAN_FOR_SPECIFIC_SERVICE, HORIZONTAL_PING_SCAN, WEB_ATTACKS,
DDOS_TO_SPECIFIC_SERVICE, FINGERPRINTING, SEND_SPAM //web_attacks_and_send_spam
```

Figure 3.19: Attack plans for evaluation in the network domain

1: The action for the apriori step is requested from the defender, based solely on the confusion matrices
2: Ask the model for an action to take
3: Generate an observation (and record it) according to the $P(o_i|a_j, d_k)$ distribution, where a_j is the actual attacker's action and d_k is the defender's action selected by the model
4: **for** each stage **do**
5: Feed the observation to the model
6: Ask the model for an action to take
7: Generate an observation (and record it) according to the $P(o_i|a_j, d_k)$ distribution, where a_j is the actual attacker's action and d_k is the defender's action selected by the model
8: **end for**
9: Evaluate the observed plan against the attacker's plan

Figure 3.20: Evaluation pseudocode

the defender action with the greatest probability of observing the actual attacker action or the smallest probability respectively. As BR, WR, Rnd and plans made exclusively from one of each of the defender's actions are independent of the observations we could easily calculate a good approximation of the estimated plan quality by averaging over a much greater number of trials.

Our way of calculating BR, WR, Rnd and Actions may not seem necessary as one could say that the expected value of the utility function can be calculated by taking the expected number of observations in all cases and then calculating the utility for that might be sufficient. However this is not possible. The reason for that is formulated in the Jensen's inequality (see [28]). Jensen's inequality states that this value would be only a lower bound of the actual expected value of the utility function. As a result we had to approximate the mean value using an average over as many iterations as possible. The BR, WR and Actions were averaged over 10^6 iterations. The Rnd evaluation has been performed by averaging over 1000 random plans with 10000 random samples for each of them.

This could not be done with the plan generated by the game theoretic approach, because different observations would affect, which defender's actions is going to be chosen next, therefore the resulting utility would be misleading. As a result, the estimations of the WR, BR and random selection are much more accurate, than that of the data collected during the experiment.

In order to compare the performance against some established method, we have given this task to CAMNEP as well. We have generated a single plan for each of the seeds, then we have evaluated its performance against the attacker plan and then evaluated its performance against the attacker's plan with the given seed the same way as we evaluated BR and WR. We realize that this is not a correct approach for the same reason as it is not correct for the game theoretic evaluation, but CAMNEP itself does not optimize the same utility as our game theoretic approach, and serves only as a point of reference. CAMNEP optimizes to minimize the rate of false positives and also works over all confusion matrices for the full duration of the attack. It does not work only with the meaningful matrices that contain the attacker's actions and does consider the possibility that the attacker's action might span over multiple stages as well and therefore is at a disadvantage.

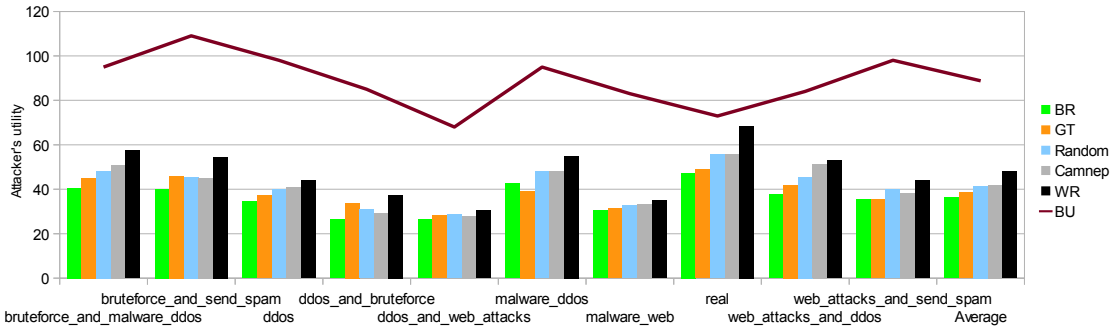
Results

In this section we are going to summarize our results. The main parts of this section are evaluations of the average performance of our game theoretic approach (GT), best response (BR), worst response (WR), random response (Rnd) and each of the possible defender's action's (Actions), if used constantly over the whole game. We have also included the Base utility, which is the sum of predicate values in the attack plan without any penalty for defender's observations. The results can be found in Figures 3.25, 3.21, 3.22. We have picked the real plan and the bruteforce_with_malware_ddos as good examples of the behaviour. The plotted results may be found in Figures 3.23 and 3.24. The results clearly show the game-theoretic defender as a good solution finder, as it usually fares much better than Rnd.

As we can see in Figure 3.25, the results seem a little odd sometime. It appears that our method sometimes beats the BR (e.g., plan bruteforce_and_send_spam, seed 3) and also loses to the WR on occasion as well (e.g., plan ddos_and_web_attacks, seed 1). This cannot be so in the long run. In the previous Section 3.8.3, we have already given the reason for this phenomenon. It is simply because of a small sample, caused by high computational time requirements of the evaluation. Much larger sample is a valid point for the future development. Still, our method performs much better on average than random and it is only slightly worse than the BR. This can be seen in Figure 3.21. In the aggregated results, the GT approach is only 2.5 utility points worse than the best response. By calculating the confidence intervals for the mean values with $\alpha = 0.05$, we have found, that

Plan	BU	GT	BR	Rnd	WR	CP
bruteforce_and_malware_ddos	95.00	45.13	40.58	48.16	57.44	50.53
bruteforce_and_send_spam	109.00	45.99	40.11	45.51	54.58	45.03
ddos	98.00	37.13	34.60	39.83	44.03	41.01
ddos_and_bruteforce	85.00	33.81	26.52	31.03	37.38	29.26
ddos_and_web_attacks	68.00	28.35	26.38	28.52	30.30	27.90
malware_ddos	95.00	39.19	42.64	48.01	54.96	48.19
malware_web	83.00	31.26	30.46	32.74	34.88	33.11
real	73.00	48.79	47.35	55.76	68.46	55.60
web_attacks_and_ddos	84.00	41.88	37.54	45.44	52.94	51.18
web_attacks_and_send_spam	98.00	35.28	35.54	39.79	43.88	38.09
Aggregated	**88.80**	**38.68**	**36.17**	**41.48**	**47.88**	**41.99**

(a) Table



(b) Plot

Figure 3.21: Results (CP refers to CAMNEP, BU to Base utility)

these are still considerably large intervals (see Figure 3.22) for all the cases and therefore these values should not be taken as precise estimates.

However, the situation is slightly better when aggregated over all plans, seeds and repetitions. The upper bound of the confidence interval is lower, than the estimated mean of Rnd. Because our estimation of the mean of Rnd quite precise (size of this confidence interval is negligible 0.0108), our method is *significantly better given the selected* α.

Confusion Matrix Filtering

In order to reduce the computational requirements of our solution, we remove the classifiers with confusion matrices very similar to some other classifiers (see Section 3.7.4). In this section, we evaluate how many of the classifiers are preserved after this filtering in the real word data used in our experiment. The histogram in Figure 3.26 shows the numbers of instances in which a particular number of actions was preserved. From the graph, we can see that we cannot infer some simple probabilistic distribution for the number of actions as they vary greatly. Very often, only one or two actions were left for the defender. This happened in over 1500 test instances. It may seem that evaluation during a stage, in which the defender has only one action, is meaningless, but we have to remember that attacker's tree gets updated as well. On the other hand, almost any numbers of

Plan	Mean	StdDev	LB	UB
bruteforce_and_malware_ddos	45.13	21.44	39.70	50.55
bruteforce_and_send_spam	45.99	23.60	40.02	51.96
ddos	37.13	12.99	33.85	40.42
ddos_and_bruteforce	33.81	18.11	29.23	38.39
ddos_and_web_attacks	28.35	13.93	24.83	31.88
malware_ddos	39.19	17.40	34.79	43.59
malware_web	31.26	16.14	27.18	35.35
real	48.79	18.72	45.05	52.54
web_attacks_and_ddos	41.88	19.31	37.00	46.77
web_attacks_and_send_spam	35.28	13.38	31.90	38.67
Aggregated	**39.25**	**18.94**	**37.78**	**40.73**

Figure 3.22: Confidence intervals (LB and UB refer to lower bound and upper bound respectively)

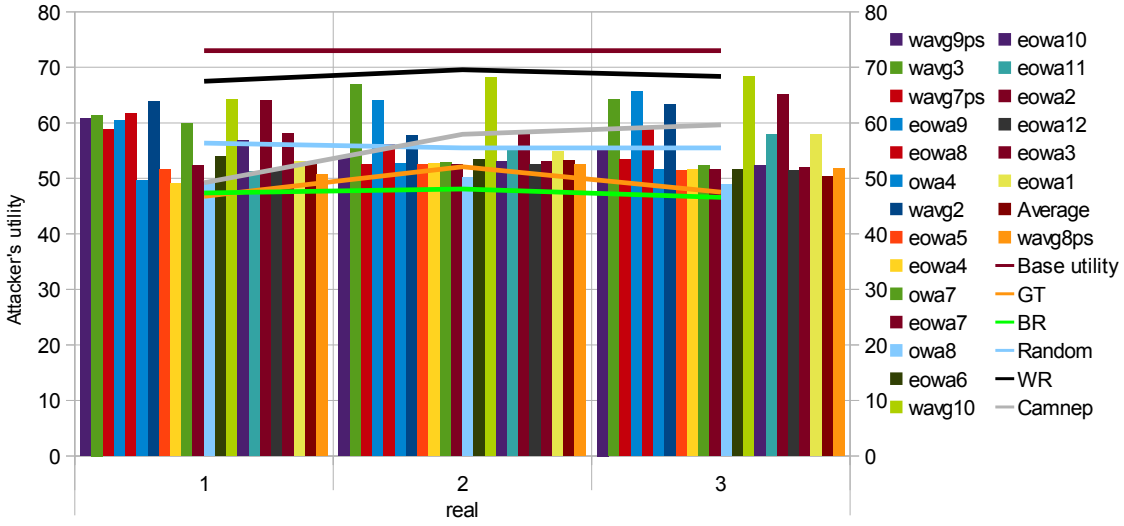


Figure 3.23: Real plan results

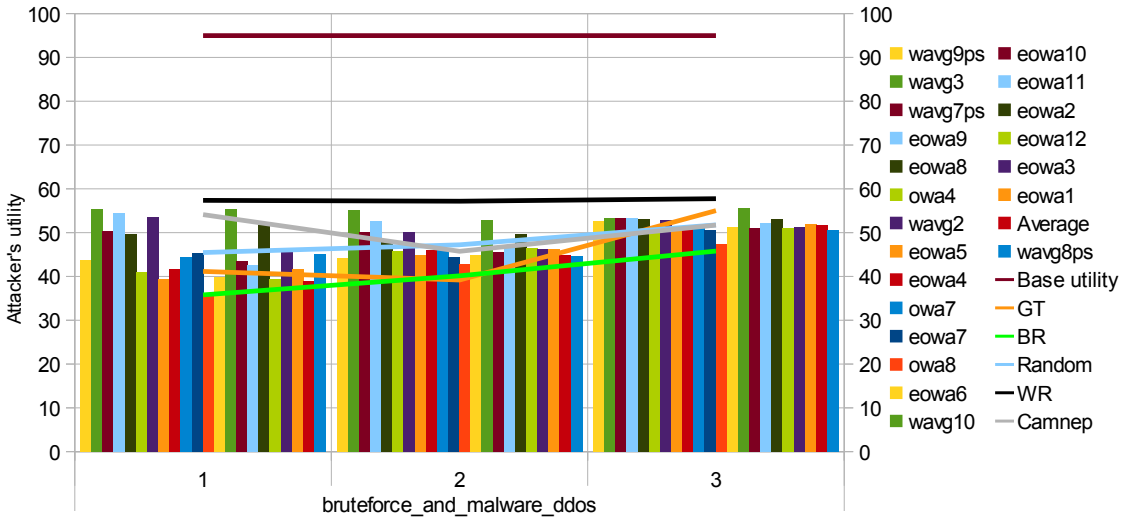


Figure 3.24: Bruteforce_and_malware_ddos_plan results

actions is preserved in some of the experiments. In 80 cases, all 22 actions were preserved.

Computational Resources

We have run the experiment on a single multi-core machine. It has an 8-core 2.4GHz Intel Xeon processor and 24GB of RAM. Four instances of the experiment ran concurrently. Each with its own dedicated core with a 100% load. The solution is single-threaded and therefore in would not really benefit from multiple cores too much. We have never crossed 4.5GB threshold for a single instance of the memory footprint during the experiment. The usual size of allocated memory space was 3.5GB.

The number of iterations (samples) of the concurrent MCTS algorithm performed in a single stage hugely varied with the number of defender's actions that have survived the filtering described in Section 3.7.4. However, the aggregated number of iterations performed per stage can bee seen in Figure 3.27. It presents a histogram of the number of instances (regardless of plan, seed or repetition) on the y-axis and the number of iterations accomplished in the fixed time period on the x-axis. The graph shows that the distribution of the iteration counts is roughly binomial/Poisson and most of the time there is roughly between ten and twenty million iterations. In most instances, the more then 10^7 iterations were performed, however, in some cases even 10^8 can be made in time.

3.8.4 Summary

We have evaluated our approach to the problem through two distinct setups – a purely synthetic and a real-world-network-based domain.

We have presented an evaluation of a synthetic setup first. It featured two games sharing the same action domain for both players and plan space for the attacker. However, they differed in their confusion matrices. The first game featured a mixed Nash equilibrium (NE), while the second

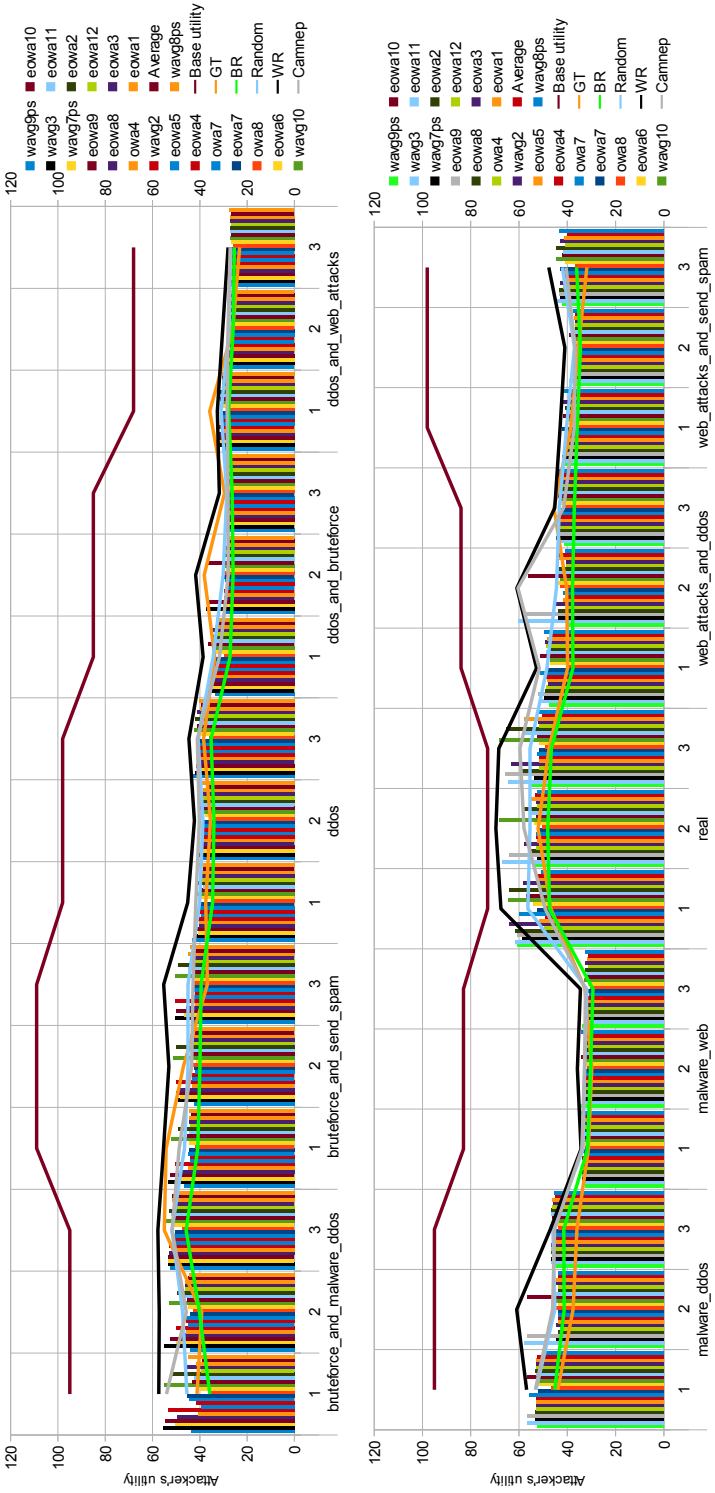


Figure 3.25: Complete results

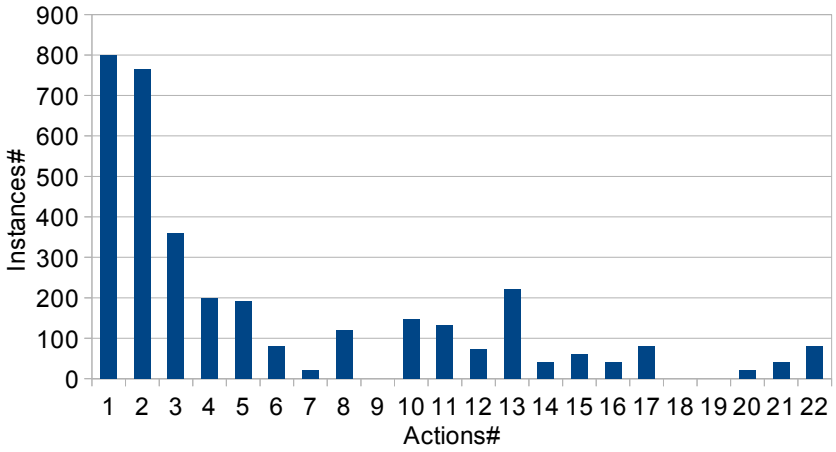


Figure 3.26: Histogram of action counts aggregated over every stage, plan, seed and repetition

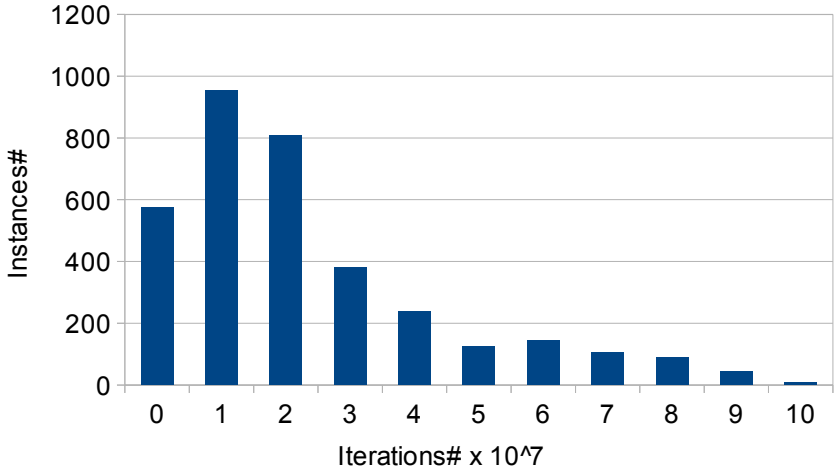


Figure 3.27: Histogram of iteration counts aggregated over every stage, plan, seed and repetition

featured a pure NE. We have seen that the calculated solutions are nearly the actual NE in both cases. This supports the hypothesis that the proposed algorithm converges to the optimal solution.

The real-network-based domain was the second setup. We have run a series of evaluations of our approach against various other solutions. We have seen, that the random response solution fared significantly worse, since our solution had the upper bound of its confidence interval with $\alpha = 0.05$ lower than the than the Rnd solution. CAMNEP, which optimizes a different criterion, performed even worse with respect to the utility used in this evaluation.

Based on the results, we conclude that the proof of concept implementation of the proposed method was successful and the next steps of this research should be relaxing the assumptions analyzed in Section 3.7.2.

Chapter 4

Conclusions and Future Work

The aim of the presented project was to investigate the potential of game-theoretical modeling of the attacker-defender interaction in the intrusion detection game. The key difference with respect to the past work in the domain is the integration of the game-theoretical methods with an actual, industry-strength anomaly detection system and operating this combination on live traffic data.

Given the results presented in Section 3.8 and elsewhere in the paper, we can conclude that the simplified model presented in Section 3.2 and the extensive game model used in the core of this work (defined in Section 3.3 and solved throughout the rest of the Chapter 3) *can be integrated with live IDS* under *reasonable*, but *very restrictive assumptions*.

In practice, we conclude that this area of research has a very high potential to produce relevant, deployable results within next 5 years, based on the trend in the computational power of current processors, average memory required for the computation and the growing sophistication of methods for efficient solving of realistic IDS games.

The critical assumptions that need to be addressed are related to the following problems:

- Time representation and time-related assumptions in the game theoretical model.
- Handling of several concurrent attackers.
- Identification of attacker coalitions, where the actions from several attackers aim to achieve a common goal.
- Sufficient detection precision and more thorough, two-way integration between the IDS and the game-theoretical model.

We argue that further, carefully designed applied research in this area should concentrate on progressive reduction of assumptions that are currently necessary to make the game theoretical model computationally solvable. The advances in game theory and gameplaying will need to be reflected in the security games domain. Further IDS and network security research is necessary to address the assumptions from the other side: by identifying the coalitions of attackers and treating them as a single individual (through the analysis of peer-to-peer networks), through more precise classification of events and through advances in the detection sensitivity and precision.

In the same time, the attackers are improving the attack techniques on their side as well, therefore ensuring that the security of future critical networks will remain an ongoing struggle.

Bibliography

[1] T. Alpcan and T. Başar. A game theoretic approach to decision and analysis in network intrusion detection. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, pages 2595–2600, Maui, HI, December 2003.

[2] T. Alpcan and T. Başar. An intrusion detection game with limited observations. In *12th Int. Symp. on Dynamic Games and Applications*, Sophia Antipolis, France, July 2006.

[3] J.Y. Audibert and S. Bubeck. Minimax policies for adversarial and stochastic bandits. In *22nd annual conference on learning theory*, 2009.

[4] Peter Auer, N Cesa-Bianchi, Y Freund, and R.E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2003.

[5] David Auger. Multiple Tree for Partially Observable Monte-Carlo Tree Search. In *Applications of Evolutionary Computation*, pages 53–62. Springer, 2011.

[6] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. Can machine learning be secure? In *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25, New York, NY, USA, 2006. ACM.

[7] Gary S. Becker. Crime and punishment: An economic approach. *The Journal of Political Economy*, 76(2):169–217, 1968.

[8] K. Borders and A. Prakash. Quantifying information leaks in outbound web traffic. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 129–140. IEEE, May 2009.

[9] M. M. Breunig, H. P. Kriegel, R. T. Ng, and J. Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, pages 93–104, New York, NY, USA, 2000. ACM.

[10] The Bro Network Security Monitor. `http://bro-ids.org/`, 2011. Accessed in June 2011.

[11] Christer Carlsson and Robert Fullér. *Fuzzy Reasoning in Decision Making and Optimization*. Physica Verlag, Springer, Heidelberg, 2002.

[12] Lin Chen and J. Leneutre. A game theoretical framework on intrusion detection in heterogeneous networks. *Information Forensics and Security, IEEE Transactions on*, 4(2):165–178, June 2009.

[13] Paolo Ciancarini and Gian Piero Favini. Monte Carlo tree search in Kriegspiel. *Artificial Intelligence*, 174(11):670–684, jul 2010.

[14] Cisco Systems. Cisco IOS NetFlow. http://www.cisco.com/go/netflow, 2007.

[15] B. Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101 (Proposed Standard), January 2008.

[16] Cognitive security. Cognitive One. `http://cognitive-security.com/`, 2011. Accessed in June 2011.

[17] C. Daskalakis, A. Deckelbaum, and A. Kim. Near-optimal no-regret algorithms for zero-sum games. In *Proceedings of the Twenty-Second Annual ACMSIAM Symposium on Discrete Algorithms*, pages 235–254, 2011.

[18] D. Denning. An intrusion detection model. In *IEEE Security and Privacy*, 1986.

[19] Dorothy E. Denning. An intrusion-detection model. *IEEE Transaction Software Engineering*, 13(2):222–232, 1987.

[20] Levent Ertoz, Eric Eilertson, Aleksandar Lazarevic, Pang-Ning Tan, Vipin Kumar, Jaideep Srivastava, and Paul Dokas. Minds - minnesota intrusion detection system. In *Next Generation Data Mining*. MIT Press, 2004.

[21] D.P. Foster, H.P. Young, et al. Regret testing: learning to play nash equilibrium without knowing you have an opponent. *Theoretical Economics*, 2006.

[22] M. Genesereth, N. Love, and B. Pell. General game playing: Overview of the aaai competition. *AI magazine*, 26(2):62, 2005.

[23] Malik Ghallab, Craig K. Isi, Scott Penberthy, David E. Smith, Ying Sun, and Daniel Weld. PDDL - The Planning Domain Definition Language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[24] J.C. Gittins, R. Weber, and K.D. Glazebrook. *Multi-armed bandit allocation indices*. Wiley Online Library, 1989.

[25] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, February 2008.

[26] Peter Haag. NFDUMP - NetFlow processing tools. `http://nfdump.sourceforge.net/`, 2007.

[27] Joseph L. Hellerstein. Why feedback implementations fail: the importance of systematic testing. In *Proceedings of the Fifth International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, FeBiD '10, pages 25–26, New York, NY, USA, 2010. ACM.

[28] J. Jensen. Sur les fonctions convexes et les inã©galitã©s entre les valeurs moyennes. *Acta Mathematica*, 30:175–193, 1906. 10.1007/BF02418571.

[29] Hilmi G. Kayacik and A. Nur Zincir-Heywood. Mimicry attacks demystified: What can attackers do to evade detection? *Privacy, Security and Trust, Annual Conference on*, 0:213–223, 2008.

[30] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo Planning. In *ECML-06*, 2006.

[31] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosis Network-Wide Traffic Anomalies. In *ACM SIGCOMM '04*, pages 219–230, New York, NY, USA, 2004. ACM Press.

[32] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining Anomalies using Traffic Feature Distributions. In *ACM SIGCOMM, Philadelphia, PA, August 2005*, pages 217–228, New York, NY, USA, 2005. ACM Press.

[33] M. Lanctot, K. Waugh, M. Zinkevich, and M. Bowling. Monte carlo sampling for regret minimization in extensive games. *Advances in Neural Information Processing Systems*, 22:1078–1086, 2009.

[34] Yu Liu, Cristina Comaniciu, and Hong Man. A bayesian game approach for intrusion detection in wireless ad hoc networks. In *GameNets '06: Proceeding from the 2006 workshop on Game theory for communications and networks*, page 4, New York, NY, USA, 2006. ACM.

[35] J. Newsome, B. Karp, and D. Song. Paragraph: Thwarting signature learning by training maliciously. In *Recent Advances in Intrusion Detection*, pages 81–105. Springer, 2006.

[36] M.J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[37] A. Parker, D. Nau, and VS Subrahmanian. Overconfidence or paranoia? search in imperfect-information games. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1045. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[38] A. Parker, D. Nau, and VS Subrahmanian. Paranoia versus Overconfidence in Imperfect-Information Games. In *Heuristics, Probabilities, and Causality: A Tribute to Judea Pearl*, pages 63–84. 2010.

[39] Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., Suite 330, 1201 5th Street S.W, Calgary, Alberta, Canada, T2R-0Y6, 1998.

[40] E. Raboin, D. Nau, U. Kuter, S.K. Gupta, and P. Svec. Strategy generation in multi-agent imperfect-information pursuit games. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 947–954. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[41] S. D. Ramchurn, C. Sierra, L. Godo, and N. R. Jennings. A computational trust model for multi-agent interactions based on confidence and reputation. In *Proceedings of 6th International Workshop of Deception, Fraud and Trust in Agent Societies, Melbourne, Australia.*, pages 69–75, 2003.

[42] Martin Rehak and Michal Pechoucek. Trust modeling with context representation and generalized identities. In *Cooperative Information Agents XI*, number 4676 in LNAI/LNCS. Springer-Verlag, 2007.

[43] Martin Rehak, Michal Pechoucek, Martin Grill, and Karel Bartos. Trust-based classifier combination for network anomaly detection. In *Cooperative Information Agents XII*, LNAI/LNCS. Springer-Verlag, 2008.

[44] Martin Rehák, Michal Pechoucek, Martin Grill, Jan Stiborek, Karel Bartoš, and Pavel Celeda. Adaptive multiagent system for network traffic monitoring. *IEEE Intelligent Systems*, 24(3):16–25, 2009.

[45] Martin Rehák, Eugen Staab, Volker Fusenig, Michal Pechoucek, Martin Grill, Jan Stiborek, Karel Bartos, and Thomas Engel. Runtime monitoring and dynamic reconfiguration for intrusion detection systems. In Engin Kirda, Somesh Jha, and Davide Balzarotti, editors, *Recent Advances in Intrusion Detection, 12th International Symposium, RAID 2009, Saint-Malo, France, September 23-25, 2009. Proceedings*, pages 61–80, 2009.

[46] Martin Rehak, Eugen Staab, Michal Pechoucek, Jan Stiborek, Martin Grill, and Karel Bartos. Dynamic information source selection for intrusion detection systems. In K. S. Decker, J. S. Sichman, C. Sierra, and C. Castelfranchi, editors, *Proceedings of the 8th International*

Conference on Autonomous Agents and Multiagent Systems (AAMAS '09), pages 1009–1016. IFAAMAS, May 2009.

[47] Benjamin I. P. Rubinstein, Blaine Nelson, Ling Huang, Anthony D. Joseph, Shing hon Lau, Nina Taft, and J. Doug Tygar. Evading anomaly detection through variance injection attacks on pca. In Richard Lippmann, Engin Kirda, and Ari Trachtenberg, editors, *Recent Advances in Intrusion Detection, 11th International Symposium, RAID 2008, Cambridge, MA, USA, September 15-17, 2008. Proceedings*, volume 5230 of *Lecture Notes in Computer Science*, pages 394–395. Springer, 2008.

[48] Benjamin I.P. Rubinstein, Blaine Nelson, Ling Huang, Anthony D. Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and J. D. Tygar. ANTIDOTE: understanding and defending against poisoning of anomaly detectors. *Internet Measurement Conference*, pages 1–14, 2009.

[49] Karen Scarfone and Peter Mell. Guide to intrusion detection and prevention systems (idps). Technical Report 800-94, NIST, US Dept. of Commerce, 2007.

[50] Mohammad Shafiei, Nathan Sturtevant, and Jonathan Schaeffer. Comparing UCT versus CFR in simultaneous games. In *IJCAI Workshop on General Game Playing (GIGA*, 2009.

[51] Y. Shoham and K. Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge Univ Pr, 2009.

[52] Sourcefire, Inc. SNORT – Intrusion Prevention System. `http://www.snort.org/`, 2007. Accessed in January 2007.

[53] Avinash Sridharan, Tao Ye, and Supratik Bhattacharyya. Connectionless port scan detection on the backbone. Phoenix, AZ, USA, 2006.

[54] Eugen Staab, Volker Fusenig, and Thomas Engel. Towards trust-based acquisition of unverifiable information. In *Cooperative Information Agents XII*, volume 5180 of *LNAI/LNCS*, pages 41–54. Springer Verlag, September 2008.

[55] O. Teytaud and S. Flory. Upper confidence trees with short term partial information. *Applications of Evolutionary Computation*, pages 153–162, 2011.

[56] David Šišlák, Martin Rehák, Michal Pěchouček, Milan Rollo, and Dušan Pavlíček. AGLOBE: Agent development platform with inaccessibility and mobility support. In Rainer Unland, Matthias Klusch, and Monique Calisti, editors, *Software Agent-Based Applications, Platforms and Development Kits*, pages 21–46, Berlin, 2005. Birkhauser Verlag.

[57] Gérard Wagener, Radu State, Alexandre Dulaunoy, and Thomas Engel. Self adaptive high interaction honeypots driven by game theory. In *SSS*, pages 741–755, 2009.

[58] Kuai Xu, Zhi-Li Zhang, and Supratik Bhattacharrya. Reducing Unwanted Traffic in a Backbone Network. In *USENIX Workshop on Steps to Reduce Unwanted Traffic in the Internet (SRUTI)*, Boston, MA, July 2005.

[59] Kuai Xu, Zhi-Li Zhang, and Supratik Bhattacharyya. Profiling internet backbone traffic: behavior models and applications. In *SIGCOMM '05*, pages 169–180, New York, NY, USA, 2005. ACM Press.

[60] R.R. Yager. On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):183–190, 1988.

[61] Zhengyu Yin, Dmytro Korzhyk, Christopher Kiekintveld, Vincent Conitzer, and Milind Tambe. Stackelberg vs. nash in security games: interchangeability, equivalence, and uniqueness. In *AAMAS '10: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pages 1139–1146, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[62] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret minimization in games with incomplete information. *Advances in Neural Information Processing Systems*, 20:1729–1736, 2008.

Appendix A

Towards Undetected Attacks

A.1 Introduction

The network intrusion detection systems (NIDS) are becoming a standard part of the security measures protecting enterprise computer networks against attacks from the outside. NIDS are usually deployed on the perimeter of the protected network (core routers), such that the ordinary users within the protected network are not aware of their existence. The role of NIDS is not limited only to detection of attacks from the outside, but they are used to detect malicious traffic generated within the protected network in order to identify hosts infected by malware [25], to detect information ex-filtration [8], and other types of unwanted traffic (p2p networks, skype, etc.).

Most frequently deployed NIDS, such as SNORT [52] or BRO [10], rely on the signature matching mechanism, where payload of packets are inspected and matched against signatures of known malware and other threats. While the signature based NIDS have low false positive rates, they posses many undesirable properties, some of them being the need of keeping database of signatures up to date, which implies the inability to detect zero day attacks (attacks which signature is not in the database), and inability to detect attacks, when the payload is encrypted. To alleviate these burdens, a lot of research efforts is devoted to intrusion detection systems based on the anomaly detection paradigm.

Anomaly detection based NIDS [18] does not use any database of signatures. Instead, it inspects the traffic on the network and identifies statistical outliers — parts of the traffic which do not conform to modeled trends or deviate from the majority. The anomalous traffic is reported with the hope that most of it is malicious or unwanted. The anomaly based NIDS relies on assumptions that (a) most of the observed traffic is benign and (b) malicious traffic will in some way deviate from the benign majority.

Although the advantages of anomaly detection based NIDS are appealing, they are not widely used in practice, since they usually suffer from higher false positive rate compared to the signature based approaches and there are concerns about their security. The knowledgeable attack can modify its attack in such a way that the intrusion detection system does not raise any alarm. Moreover, as the NIDS learns the model from the observed traffic, the attacker with sufficiently large resources (obtained for example by hiring botnet network) can attack the learning algorithm itself and force it to learn to recognize malicious traffic as the legitimate one [6, 48].

In this work, we practically investigate the security of NIDS based on anomaly detection paradigm

against knowledgeable attacker. The goal is to perform undetected attack against the network monitored, under worst-case scenario for the defender (NIDS), which has full access to all internal states of the system. Although this scenario of fully-knowledgeable adversary is unlikely to happen in practice, [1] it is important for the evaluation of the security and limitations of the system.

The experiments are performed against CAMNEP intrusion detection system [16], which uses ensemble of six diverse yet simple anomaly detectors [58, 20, 53, 32]. The results shows that while it is possible to bypass individual CAMNEP's detectors, bypassing all of them simultaneously is either impossible, or the strength of the attack is decreased below an acceptable level. This suggests that modification of the internal state of the IDS, as has been demonstrated in prior art against one detector [48], might be very difficult, or even impossible, if NIDS uses ensemble of diverse detectors simultaneously.

The rest of the chapter is organized as follows. Section A.2 summarizes the prior art and different approaches towards realization of an undetected attack. Section A.3 details parts of CAMNEP NIDS important for this chapter, namely the anomaly detectors. The very same section also contains description of evasion strategies against them and discuss the implications. Section A.4 describes the experimental details with the algorithm generating the actual attacks. The experiments are presented in Section A.5. Finally, the chapter is concluded in Section A.6.

A.2 Cloaking the attack

Ptacek et al. [39] has identified two, complementary strategies to evade the detection. While *evasion* strategy decreases the intensity of the attack or modifies it, *insertion* strategy add supplemental traffic to mask the actual traffic caused by the attack. Both strategies are general and can be applied to any type of IDS system. Both of them are discussed in detail below in the chosen application domain, which is network intrusion detection.

A.2.1 Evasion — lowering the intensity

The first attacker's strategy is to **lower the intensity** of the attack and spread it over longer period of time. The rationale behind is that by making the intensity of the attack very small, it will be more difficult to separate it from the benign traffic. Moreover, as will be discussed in Section A.3, some detectors inspect only sources with number of flows exceeding certain threshold. These facts support the belief that this strategy might almost always work. Its drawback (from the point of view of the attacker) is that the intensity of the attack can be so low that the possible reward for the attacker would not be interesting anymore, and he will look for other target. An example is brute-force cracking of SSH password, where having only couple trials per minute is practically useless.

A.2.2 Insertion — generating additional flows

In the second strategy, the attacker **generates additional traffic** that is not directly related to the attack. Its purpose is to conceal the actual attack, such that the overall statistics of attacker's traffic observed by the detection system look innocuous.

[1] Even though the attacker can monitor network at the same point as the system he is attacking, which enables him to run his own CAMNEP to get the access to the internal state of the system, his knowledge would not be complete. The internal state of CAMNEP is randomized through the challenge mechanism, which causes internal states of systems fed with the same inputs deviates from each other.

Barreno et al. [6] further proposes to use supplemental traffic to manipulate the internal state of the detector, such that attacks will be considered as a benign traffic. He specifically challenges IDS systems based on machine learning principles. The reward is that even attacks of high strength can be invisible.

The idea has been experimentally verified by Rubinstein et al. [48], who has targeted the detector of Lakhina et al. [32]. Similar in essence, but simpler approach has been investigated by Newsome et al. [35], who proposed to shift the detection threshold toward accepting the malicious traffic by adding false positives[2]. Both works [48] and [35] exploits the fact that targeted systems had distinct training phase (during which the model of the traffic is inferred) and detection phase (during which the system is actually used to detect the attacks). It makes the situation easier for the attacker, since alarms during training phase, which are highly likely, are not investigated. In the CAMNEP system challenged in this work, the observed traffic is simultanously scrutinized for attacks and used for adaptation of the IDS to the current state of the network. This makes the situation more difficult for the attacker, since he should not raise any alarm.

Although the insertion strategy is very interesting, because it possibly enables to perform large scale attacks, there are several caveats to be aware of. First, the amount of the traffic that has to be added to cover the attack might be so large that the attacker cannot generate it with his limited resources. Second, the total volume of the traffic (attack flows + cover flows) can be so large that it will be easily detectable by detectors monitoring volume of the traffic [32, 20]. Third, the additional traffic might disturb some network statistics the attacker is not aware off [3], which can make his activities easily detectable.

Here we point out that the problem of bypassing the detector is very similar to steganography, where communicating parties try to hide the secret message into innocuous looking objects (e.g. digital image). In the steganography domain it has been already many times experimentally verified that making more embedding changes, which corresponds to the insertion strategy, increases the probability of being detected. From these reasons, we consider the evasion strategy more important. The attacker has higher probability of being successful with less resources. Consequently, this strategy represents higher risk and it is the focus of this chapter. The insertion strategy is left to the future work.

A.3 CAMNEP IDS

In this section, we describe those parts of CAMNEP NIDS that are most important for our work. The input to the CAMNEP system are statistics of flows (number of bytes, number of packets, duration, etc.) in the NETFLOW format [14] aggregated over five-minute long time windows. By the flow it is understood a unique connection between two computers determined by the five-tuple (sIP, sPrt, dIP, dPrt, protocol ID). CAMNEP uses ensemble of six different detectors [58, 20, 53, 32] to decide, which flows are malicious and which are benign. The detectors are diverse, as some of them are based on heuristic principles while others rely on machine learning algorithms in order to adapt to the current traffic on the network. In order to make the text self-contained and explain the modifications with respect to prior art, the detectors are described below together with the evasion strategies to bypass them.

[2] His idea is to raise so many false alarms, that the network operator will turn off the system
[3] This is actually inconsistent with our assumptions, but yet worth to note

A.3.1 Xu sIP and Xu dIP

Two heuristic based detectors Xu sIP and Xu dIP are inspired by the original work of Xu et al. [58]. *Xu sIP* calculates relative uncertainty[4] of source and destination ports, and destination IP addresses of all flows originating from a given source IP address. If all three quantities are within the range $(0.20, 0.80)$, than all traffic originating from this IP address is deemed to be benign. The *Xu dIP* variant is essentially the same, except the relative uncertainnesses are calculated with respect to destination IP addresses.

Both detectors use fixed static rules that do not adapt to the background traffic. According to our measurements (not presented here), they are good at detecting horizontal scans and brute force cracking of SSH passwords.

In order to bypass the detectors, the attacker needs to modify ranges of IP addresses and ports of attack flows such that their relative uncertainty falls within the range $(0.20, 0.80)$. This can be easily achieved by increasing or decreasing ranges of appropriate quantities (e.g. the attack flows spans over more source ports, less source IP addresses, etc.). Notice that the relative uncertainty depends on the distribution of flows, not on their number. This property simplifies the evasion strategy. In the presented experiments it was always possible to modify the attack to bypass both detectors individually.

A.3.2 TAPS

TAPS detector is an implementation of the scan detector presented in [53]. It deems the source IP address being malicious, if it has flows with either high ratio of destination IP address to destination ports, or high ratio of destination ports to destination IP addresses.

Although these two ratios can provide bounds that the attack flows have to meet, there is an other way. As TAPS is designed to detect scans, it evaluates only flows with one packet. Consequently, to bypass TAPS detector it suffice that all attack flows have at least two packets — such flows are considered as benign by TAPS.

A.3.3 MINDS

The *MINDS* detector is a simplification of the detector proposed in [20]. For every flow, it uses following four quantities calculated over five-minute long window:

- number of flows arriving to the same destination IP address
- number of flows originating from the same IP address
- number of flows arriving to the same destination IP address from the same source port
- number of flows originating from the same IP address and targeting the same destination port

[4]The relative uncertainty is in its essence a normalized entropy. Let's assume that we want have a discrete random variable x, taking values $\{1, \ldots, k\}$. We estimate probabilities $\{p_1, \ldots, p_k\}$ from observations, such that $p_i = \frac{m_i}{m}$, where $m = \sum_{i=1}^{k} m_i$, and m_i denotes how many times i^{th} event has occurred in our observations, and the empirical estimate of the entropy of x is equal $H(x) = -\sum_{i=1}^{k} p_i \log p_i$. The relative uncertainty of x is equal $RU(x) = \frac{H(x)}{\log \min(k,m)}$.

These quantities are individually compared to the same quantities calculated in the previous time window (if there is no history, default values are provided). The degree of anomaly of a given flow is determined as a weighted average of differences between quantities from the time window t and the previous time window $t-1$.

To implement the evasion strategy, the history (values from the previous time window) provide us bounds on the number of flows, the attack has to meet. But as the detector uses only absolute number of flows, not any other statistics, it is easy to decrease the strength of the attack to fit it within the bounds. Indeed, the experiments confirm that the detector has a significant impact on the strength of attacks.

The CAMNEP's implementation significantly simplifies the prior art. First, it observes only features aggregated by the time window. Second, it does not use the advanced anomaly detection through local outlier factor algorithm [9]. These simplifications were introduced, because as has been pointed out in the original publication [20], the detector does not scales well with the number of flows.

A.3.4 Lakhina Entropy and Lakhina Volume

The original Lakhina's detectors, as described in [32], are designed to detect anomalies in the backbone traffic, where the statistics is acquired from *multiple points*. Since the CAMNEP acquires statistics about the network only from *single point*, the method is slightly modified to meet the application constraints.

Both detectors capture state of the traffic on the network in the time window t in a vector $x^t \in \mathbb{R}^{3\times n_t}$, where n_t is number of source IP addresses with more than 100 flows (we call them active) in the time window t.

In Lakhina Volume detector number of flows $\mathrm{N}_f^t(\iota)$, number of packets $\mathrm{N}_p^t(\iota)$, and number of bytes $\mathrm{N}_b^t(\iota)$, of all active source IP addresses $\iota \in \{1,\ldots,n_t\}$ are arranged in one (row) feature vector x_{v}^t. The feature vector x_{v}^t corresponding to the time window t is equal to

$$x_{\mathrm{v}}^t = \left(\mathrm{N}_f^t(1), \mathrm{N}_p^t(1), \mathrm{N}_b^t(1), \ldots, \mathrm{N}_f^t(n), \mathrm{N}_p^t(n), \mathrm{N}_b^t(n)\right)$$

Similarly, in Lakhina Entropy detector entropy of source ports $\mathrm{H}_{\mathrm{sPr}}^t(\iota)$, entropy of destination ports $\mathrm{H}_{\mathrm{dPr}}^t(\iota)$, and entropy of destination IP addresses $\mathrm{H}_{\mathrm{dIP}}^t(\iota)$ of all active source IP addresses $\iota \in \{1,\ldots,n_t\}$ are arranged in one (row) feature vector x_{e}^t. The feature vector x_{e}^t corresponding to the time window t is equal to

$$x_{\mathrm{e}}^t = \left(\mathrm{H}_{\mathrm{sPr}}^t(1), \mathrm{H}_{\mathrm{dPr}}^t(1), \mathrm{H}_{\mathrm{dIP}}^t(1), \ldots, \mathrm{H}_{\mathrm{sPr}}^t(n), \mathrm{H}_{\mathrm{dPr}}^t(n), \mathrm{H}_{\mathrm{dIP}}^t(n)\right) \tag{A.1}$$

Since both detectors process feature vectors in the same way, the anomaly detection algorithm is described below on the general row vector x^t.

CAMNEP's implementation uses feature vectors $x^{t-5},\ldots,x^{t-1}$ from previous five time windows to build the model of the traffic on the network. The model is build by means of principal component analysis (PCA), which transforms the original space, $\mathcal{X}$, where x lives into transformed space $\mathcal{Y}$, where the transformed coordinates of x are not correlated.

The PCA model is calculated as follows. The row feature vectors from previous five time windows

$x^{t-5}, \ldots, x^{t-1}$ are arranged in the matrix[5]

$$\mathbf{X} = \begin{pmatrix} x^{t-1} \\ \vdots \\ x^{t-5} \end{pmatrix} \in \mathbb{R}^{5,N}.$$

If $\bar{\mathbf{X}}$ denotes the centered version of $\mathbf{X}$ (mean of its columns of is equal to zero), then k^{k} principal component y_k is calculated as

$$y_k = \arg \max_{w:\|w\|=1} \left\| \bar{\mathbf{X}} \left(\mathbf{I} - \sum_{i=1}^{k-1} y_i y_i^{\mathrm{T}} \right) w \right\|.$$

Components $\{y_k\}_{k=1}^{N}$ are ordered according to their variance, which means that the first components has highest variance, and consequently contains the most information about $\{x^{t-5}, \ldots, x^{t-1}\}$.

Lakhina's detectors rely on the assumption that the subspace spanned by the first K principal components $\mathbf{Y}_{1:K} = (y_1, \ldots, y_k)$ is the normal traffic subspace. It has projection matrix $\mathbf{P}_{1:K} = \mathbf{Y}_{1:K}\mathbf{Y}_{1:K}^{\mathrm{T}}$. Consequently, the residual $N-K$ subspace $\mathbf{Y}_{K+1:N} = (y_{K+1}, \ldots, y_N)$ with projection matrix $\mathbf{P}_{K+1:N} = \mathbf{I} - \mathbf{P}_{1:K}$ contains the anomalous traffic. In the prior art and in the CAMNEP, the dimension of normal subspace is set to $K = 4$.

The traffic feature vector x^t from the current time window is decomposed into the part modeled by the normal subspace $\bar{x}^t$ and by the anomalous subspace $\tilde{x}^t$. It holds that

$$x^t = \bar{x}^t + \tilde{x}^t \tag{A.2}$$

$$\bar{x}^t = \mathbf{P}_{1:K} x^t \tag{A.3}$$

$$\tilde{x}^t = \mathbf{P}_{K+1:N} x^t = (\mathbf{I} - \mathbf{P}_{1:K}) x^t \tag{A.4}$$

Now, if the anomalous part $\tilde{x}^t$ contains elements in absolute value exceeding a design threshold, than IP addresses corresponding to these elements are deemed to be anomalous (malicious).

Although the Lakhina's detectors seems to be complicated, the rationale behind them is simple. They set bounds on the modeled quantities according to the current state on the network. If there is an IP address with quantities out of these bounds, it is deemed as anomalous. Consequently, the evasion strategies to bypass these detectors are similar to the strategies used for the detectors MINDS and Xu.

To evade Lakhina Entropy detector, we use the PCA model to calculate bounds on the entropies of source and destination ports, and destination IP addresses. Then, we modify distribution of the attack flows to meet them, which is exactly what is done in case of Xu detectors for relative uncertainty (remember that relative uncertainty is just a normalized entropy).

To evade Lakhina Volume detector, we do the same. Once we calculate the bounds on number of flows, bytes, and packets originating at one IP address, we modify the strength of the attack to meet the criteria. Again, this strategy is very similar to the strategy used for MINDS detector.

A.3.5 Discussion

From the above elaboration of detectors and evasion strategies, we can conclude that it is always possible to modify the attack of *any strength* such that it will bypass individual detectors based

[5]Because feature vectors from different time windows might have different dimensions, zero values are used to fill in values corresponding to inactive IP address. This mechanism ensures that all feature vectors $x^{t-5}, \ldots, x^t$ has the same dimension N

on entropies (Xu sIP, Xu dIP, and Lakhina Entropy). On the other hand, detectors observing the volume of the traffic (MINDS and Lakhina Volume) actually put constraints on the strength of the attack, which needs to be decreased to meet the criteria. The experiments, presented later confirms that.

The situation starts to be interesting, when all detectors coordinates together. As each detector (even the entropy based ones) requires the attack traffic to be modified in different way, it might happen that it is impossible to modify the attack to conform all detectors. Again, this phenomenon has been observed in experiments in Section A.5

The conclusions show that the evasion strategy, investigated here, has higher chance for success than the insertion strategy. The insertion strategy increases the overall volume of the traffic by adding cover flows that are not directly related to the traffic. Consequently, the probability of raising alarm by detectors monitoring volume of the traffic (Minds and Lakhina Volume) is higher.

A.4 Experimental details

From the analysis in the previous section it is clear that the evasion strategy has a higher chance for success. The insertion strategy is harder to implement and the danger of triggering the alarm is higher. From these reasons, this chapter presents our results on implementing the evasion strategy, and the insertion strategy is left to the future work.

As was already mentioned in the introduction, we assume the worst case scenario (for the defender), where the attacker has a *full access* to all internal states of the CAMNEP NIDS. We also assume that the attacker has limited resources in the sense that the attack is performed only from one subnet (i.e the source IPs of attack can differ only in the last octet). Again, scenarios, where the attacker controls large number of computers (e.g. by using computers infected by botnets) from different subnets to perform coordinated attack are left to future works.

To simplify the implementation, the attacks are simulated by generating the attack flows within the CAMNEP system and mixing them with a real traffic. We believe that the simulation does not have a significant impact on the credibility of the results.

Due to the time continuity, attack flows to be used in the time window $t+1$ needs to be prepared in the time window t. The attack flows are created such that they are not detected by CAMNEP's detectors at the time window t, which means that the degree of anomality on the attack flows is below the threshold α_0 (design threshold specified by the attacker). Notice that due to the time continuity, the attacker cannot be absolutely certain that the attack flows prepared in the time window t will be undetectable in the time window $t+1$, since the internal states of detectors and the background traffic in both time windows are different. But due to high temporal correlation of flow statistics, the attacker can expect that flows undetectable at the time window t will remain undetectable at the time window $t+1$.

The attack flows corresponding to a given attack are described by the following properties:

- range of source and destination IP addresses,
- range of source and destination ports,
- range of bytes per flow,
- range of packets per flow,

- total number of flows,
- layer 4 protocol ID (UDP, TCP, ICMP).

The experiments presented in Section A.5 focus on three common types of attack: the horizontal and vertical scans, and the brute force cracking of SSH password. These attacks have been chosen not only due to their ubiquity, but also because their characteristics, are very different from each other. The characteristics are listed below.
Horizontal scan:

- wide range of destination IPs,
- one destination port,
- low bytes and packets count per flow.

Vertical scan:

- one destination IP,
- wide range of destination ports,
- low bytes and packets count per flow.

Brute force cracking of SSH password:

- one destination IP,
- one destination port.

The characteristics, which are not specified for a given type of the attack are free and the algorithm generating attack flows can manipulate them in order to make the flows undetectable. The algorithm also verifies that the attack does not lose its main characteristics. For example, a brute-force cracking of SSH password whose destination port range needs to be increased from one to a thousand is not a SSH cracking any more. Similarly, changing destination IP address of vertical scan does not make sense either. To avoid this, we set acceptable bounds on properties (depending on the type of the attack) that the attack must meet in order to keep its original characteristics.

The attack flows are created by the following iterative algorithm:

1. Create the initial set of attack flows according to the specification.
2. The generated flows are passed to modified anomaly detectors asserting their detectability. If the level of anomaly exceeds threshold α_0, the detectors reports, why it happens and recommends the modification (e.g. to increase / to decrease entropy of source ports, to decrease number of flows from / to IP address, etc.).
3. If none of the detectors raised alarm on the flows, the generation is finished. Otherwise, the flows are modified as suggested by most detectors and the algorithm proceeds back to step 2.

The attack generation loop iterate unless either the attack flows meet all criteria (it is not detected and the attack did not lose its properties), or the loop exceeds certain number of iterations. In the latter case, it is assumed that the attack meeting given criteria, current state of the IDS, and the current state of the observed network is impossible.

In all experiments presented in Section A.5, the initial strength of the attack is set to twice the strength of the attack from previous time window. If the attack in the previous time window was not successful, the strength is set to 150 flows per five-minute long time window. By adopting this simple strategy, it is possible to reach the maximum strength of the attack within few time windows.

Once the attack flows are generated, they are mixed with the observed traffic in the next time step. If it is not possible to construct an undetectable attack, the system does not insert any flows and it is signaled that the generation of attack flows has failed.

The strength of the generated attack is controlled by a threshold, α_0, on the anomaly level signaled by detection agents on attack flows. The algorithm responsible for generating the attack flows always try to create attack of maximum strength (number of flows) without exceeding the threshold α_0.

A.5 Experiments

In this section, we experimentally evaluate the success of the attacker in bypassing the CAMNEP IDS. For all experiments, we have used traffic captured from the university network with approximately 25 000 flows per five-minute time window as a background traffic. As we are interested only if flows simulating attacks are detectable, we do not need the traffic to be labeled.

In Section A.3, we have emphasized that the detectability and the strength of the attack depends on the state of the IDS and on the background traffic. We can expect that during working hours, when the network is very busy, the attack will more likely to be hidden then at night, when the traffic on the network is relatively low. To investigate these effects, the used network capture spans full 24 hours (300 five-minute time windows). The attacks were designed such that the degree of anomaly was below $\alpha_0 = 0.1$ on 0–1 scale. The attacks were initiated after 50 minute long warm-up period, which allows the CAMNEP system to reach its working conditions.

A.5.1 Vertical scan

Figure A.1 shows the evolution of the strength of vertical scan attack, measured as a percentage of number of attack flows in the total number of flows in the background traffic. We have first experimenting with bypassing individual detection algorithms, and then with bypassing all of them acting simultaneously (denoted by caption the label "all"). We can observe that strength of the attacks against Xu sIP, Xu dIP, Lakhina Entropy, and TAPS detectors can grow indefinitely at exponential rate in our testing environment. From this reason, we stopped the experiment for these four agents after processing 20 five-minute long snapshots (10 snapshots for warm-up and 10 for generating the attack). Note that this is on par with our theoretical elaboration made in Section A.3, as first three agents uses entropy measures to assess the level of anomalousness.

On the other hand, attacks bypassing Lakhina Volume and Minds detectors reach their maximum strength after short period with average strength 6.23%, 0.72% respectively, of the volume of the background traffic (this corresponds to 1 321, 172, respectively flows per 5-minute long time

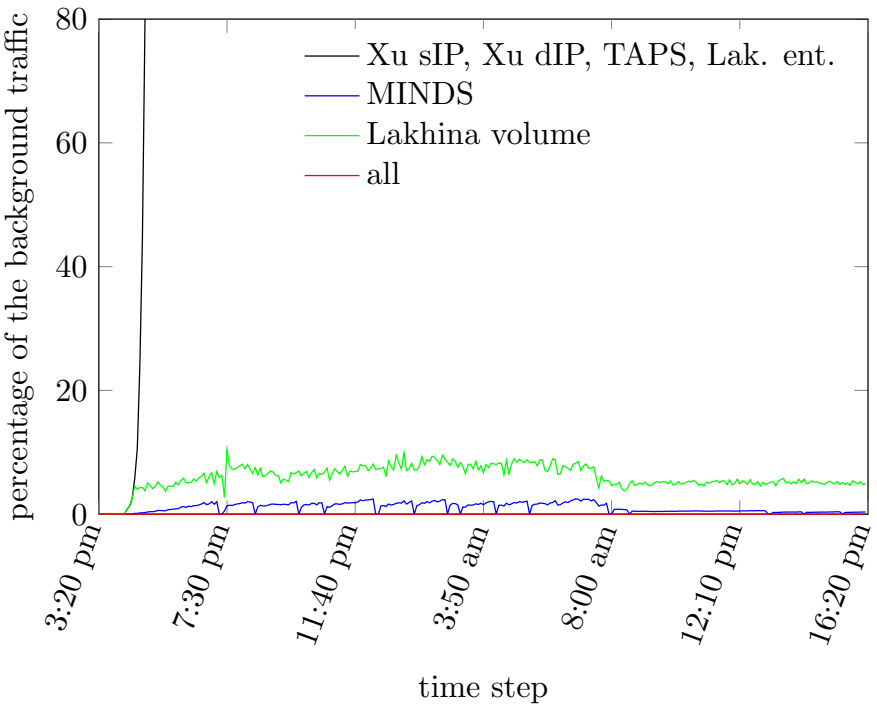


Figure A.1: Evolution of the strength of the vertical scan attack targeted to bypass individual detection agents, and all agents collaborating together (label "all"). The strength of the attack is measured as a percentage of total number of flows in background traffic.

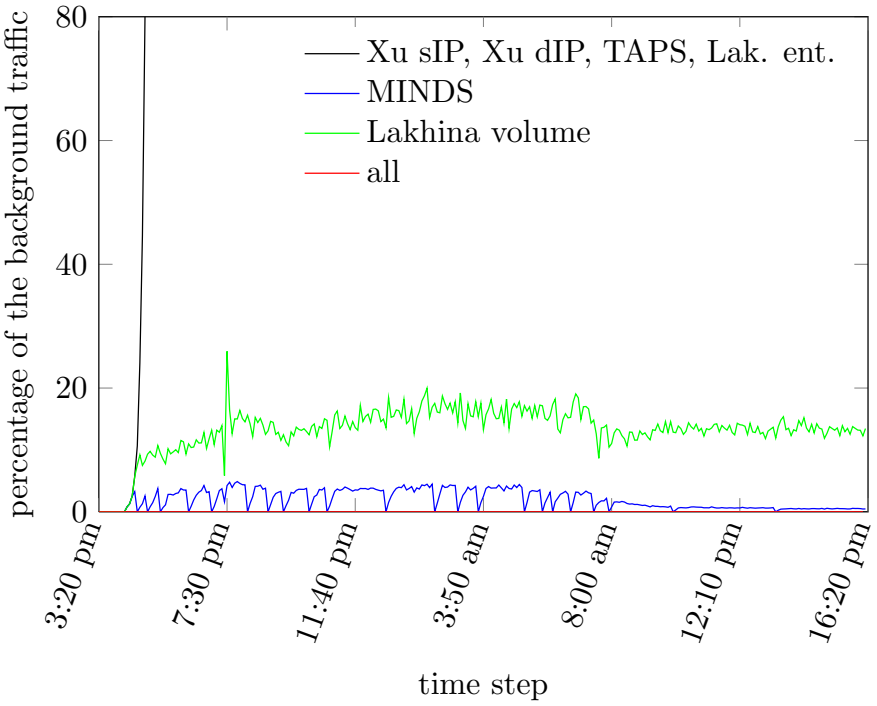


Figure A.2: Evolution of the strength of the horizontal scan attack targeted to bypass individual detection agents, and all agents collaborating together (label "all"). The strength of the attack is measured as a percentage of total number of flows in background traffic.

windows). The Minds detector seems to be the most difficult to be bypassed individually, as the attack strength was the lowest and sometimes, it was impossible to generate the attack flows (the attack was created with rate of success 93% (see Table A.1).

The graph on Figure A.1 further reveals that the strength of attacks changes with time (the time period of our capture is from 3:20pm to 4:15pm the next day). It is interesting to see that the attack against Minds agent has its maximum strength during night hours, and from approximately 7:00am, the strength declines. Although these observations contradicts our initial assumption that attacks are easier to detect in night hours, when the traffic on the network is relatively low, then during busy hours, they can be easily explained. During day hours, there is an additional traffic interfering with the attack. The MINDS detector checks that the absolute number of flows does not exceeds the threshold. Thus, if we sum background and attack traffic together, the space left for the attack is smaller during day hours. The same holds for the Lakhina Volume detector.

A.5.2 Horizontal scan

Figure A.2 shows the evolution of the strength of horizontal scan attack. Although the undetectable strength is approximately two times larger that the undetectable strength of vertical scan attack discussed in the previous section, the conclusions are pretty much the same. It is possible to launch attacks of exponentially increasing strength bypassing Xu sIP, Xu dIP, Lakhina Entropy, and TAPS detectors. Similarly, attacks bypassing Lakhina Volume and Minds detection agents are possible, but their strength is limited to 13.7% and 2.06%, respectively, of background traffic (this

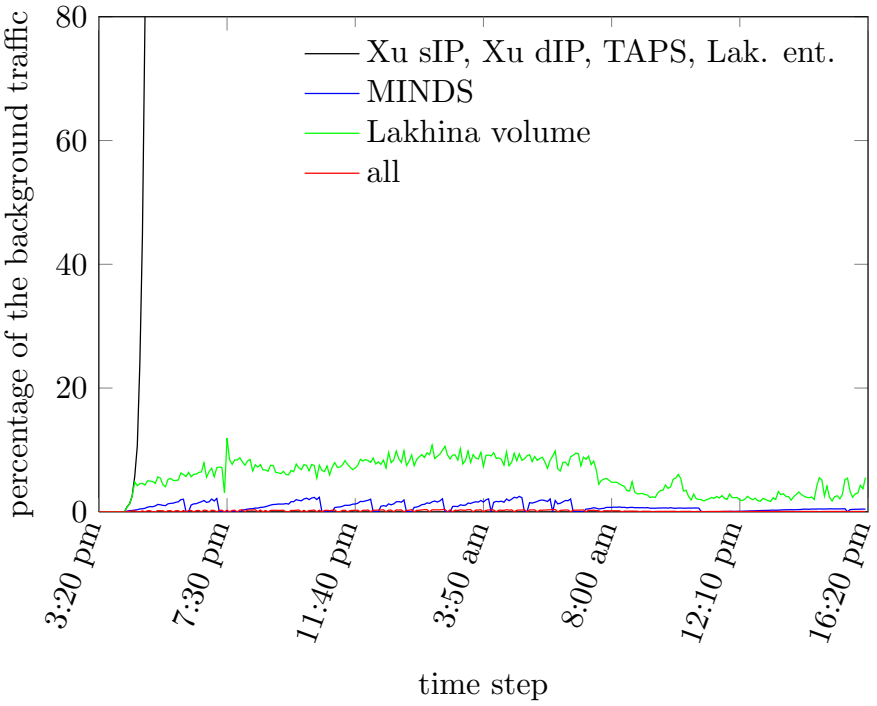


Figure A.3: Evolution of the strength of the brute-force cracking of SSH password targeted to bypass individual detection agents, and all agents collaborating together (label "all"). The strength of the attack is measured as a percentage of total number of flows in background traffic.

corresponds to 3 155, 313 respectively, flows per 5-minute long time windows) .

A.5.3 Brute-force cracking of SSH password

The very same, as has been seen in the cases of planting undetectable horizontal and vertical scans holds for brute-force cracking of SSH passwords shown on Figure A.3. Again, Xu sIP, Xu dIP, Lakhina Entropy, and TAPS detection agents can be bypassed at anomaly rate below 0.1, and Lakhina Volume and Minds detection agents can be bypassed, but with considerably lower strength of attack, which is on average 5.89% and 0.85%, respectively, of background traffic (this corresponds to 1 021, 141 respectively, flows per 5-minute long time windows)

A.5.4 Power of plenty

In Figures A.1, A.2, A.3, and in Table A.1 we can also observe strength of attacks designed to simultaneously bypass all detectors acting together (caption "all"). This means that the level of anomaly of attack flows of all detectors has to be below $\alpha_0 = 0.1$. We can see that bypassing all detectors cooperating together is more difficult than bypassing just one detector, which has been discussed above. In fact, the algorithm described in Section A.3 was not able to generate attack flows performing horizontal and vertical scans. Although it was possible to generate attack flows performing SSH brute force password cracking, the strength of the attack was only 0.14% of the background traffic (35 flows per five-minute). Taking into the account that brute force cracking of

Detector	horizontal scan		vertical scan		SSH bruteforce	
	strength	success	strength	success	strength	success
Lakhina Volume	13.7%	100%	6.23%	100%	5.89%	100%
Lakhina Entropy	$+\infty$	100%	$+\infty$	100%	$+\infty$	100%
MINDS	2.06%	92.36%	1.09%	93.05%	0.85%	85.06%
TAPS3D	$+\infty$	100%	$+\infty$	100%	$+\infty$	100%
Xu sIP	$+\infty$	100%	$+\infty$	100%	$+\infty$	100%
Xu dIP	$+\infty$	100%	$+\infty$	100%	$+\infty$	100%
All	0	0%	0	0%	0.14%	79.16%

Table A.1: The average strength (in percents of background traffic) of attacks generated to bypass detection agents at anomaly level $\alpha_0 = 0.1$. Column strength enumerates rate of successfully creating attack below the detection threshold.

password usually requires very large number of trials, having only 7 trials per minute makes the threat from SSH brute force attack virtually harmless.

A.6 Conclusion and future work

The goal of this chapter was to investigate, if the knowledgeable attacker can modify his attack, such that intrusion detection systems based on anomaly detection paradigm will not raise any alarm. The presented scenario captured the worst case point of view of the defender (IDS), where the attacker has full access to all internal states of the IDS system and it has the same view on the monitored network.

The chapter elaborated two complementary strategies to fool the IDS. The evasion strategy tries to decreases the strength of the attack. On the other hand, the insertion strategy adds supplemental flows unrelated to the attack to either conceal the attack flows, or to modify the internal states of the IDS. It has been explained, why the first strategy is more dangerous and why it is more likely to happen in practice.

We have implemented the evasion strategy against commercially available CAMNEP network intrusion detection system, which employs six detectors based on anomaly detection paradigm. The experimental results has confirmed that it is indeed possible to construct attacks bypassing individual detectors. But creating attack flows bypassing all detectors acting together has proved to be more challenging. In fact, our algorithm used to generate the attack failed to perform undetectable horizontal and vertical scans. Although the algorithm managed to generate undetectable cracking of SSH password, the strength of the attack was so low, that the attack was practically unusable.

The results emphasize the importance of ensemble based approach towards intrusion detection system, as it has been experimentally proved that bypassing one detector is easy, but bypassing many of them is very difficult, even impossible.

Moreover, the results suggest that manipulating the internal states of ensemble based IDS might be very difficult to achieve in practice. The additional traffic needed to manipulate internal state of one detector might raise alarm of the other detector. Consequently, the attacker needs to manipulate internal states of all detectors simultaneously, or manipulate one and avoid the detection of others. Although this seems to be very difficult, this approach represents a threat and it is our plan to investigate it.

We also note that the presented scenario was the worst case for the defender, as the attacker had access to all internal states of the target system and it had the same view on network. We would like to investigate more realistic scenario, where the attacker runs his own version of CAMNEP with different, only partially overlapping view on the network. In this case, the attacker does not have access to the internal state of the target system and uses his own to generate the attack. As both systems have different internal states, the generated attack might be detectable by the target system, but this remains to be verified.

Appendix B

List of Publications

In the following, we will present the the key accepted and submitted papers relevant to the present effort.

- **SSCI 2011**: Martin Rehák, Jan Stiborek, Martin Grill: *Intelligence, not Integration:Distributed Regret Minimization for IDS Control*. In Computational Intelligence in Cyber Security (CICS), 2011 IEEE Symposium [CD-ROM]. Piscataway: IEEE, 2011, p. 217-224. ISBN 978-1-4244-9905-2.
- **AAMAS 2011**: Martin Rehák, Michal Pěchouček, Martin Grill, Jan Stiborek, Karel Bartoš: *Game Theoretical Adaptation Model for Intrusion Detection System*, to appear in Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems - Innovative Applications Track (AAMAS 2011), Tumer, Yolum, Sonenberg and Stone (eds.), May, 2-6, 2011, Taipei, Taiwan.
- **IAT 2011**: Martin Rehák, Jan Stiborek and Martin Grill: *On the Value of Coordination in Distributed Self-Adaptation of Intrusion Detection System*. In Proceedings Web Intelligence and Intelligent Agent Technology WI-IAT11. Los Alamitos, CA: IEEE Computer Soc., 2011. ISBN 978-0-7695-4513-4.
- **IWCMC 2011**: Karel Bartoš, Martin Rehákand Vojtech Krmíček: *Optimizing Flow Sampling for Network Anomaly Detection*, to appear in Proceedings of the TRaffic Analysis and Classification (TRAC) Workshop (IWCMC2011-TRAC), July 2011, Istanbul, Turkey
- **SPI 2011**: Martin Rehák, Karel Bartoš, Martin Grill, Pavel Jisl, Jan Jusko, Tomas Pevný, Michal Svoboda a Jan Stiborek: *Strategic Self-Organization Methods for Intrusion Detection Systems*, to appear in Proceedings of Security and Protection of Information, May 2011, University of Defence, Brno

On the Value of Coordination in Distributed Self-Adaptation of Intrusion Detection System

Martin Rehák
Czech Technical University
Prague, Czech Republic
martin.rehak@agents.felk.cvut.cz

Martin Grill
Czech Technical University
Prague, Czech Republic
grill@agents.felk.cvut.cz

Jan Stiborek
Czech Technical University
Prague, Czech Republic
stiborek@agents.felk.cvut.cz

***Abstract*—We present an empirical study of distributed adaptation in an Intrusion Detection System. The adaptation model is based on a game-theoretical approach and we use regret minimization techniques to find globally robust behavior. We compare the effectiveness of global optimization, when all system components adopt the globally optimized strategy in a synchronized manner, with a fully distributed approach when two layers in the system adapt their strategies as a result of local adaptation process, with no synchronization or signaling. We show that the use of regret minimization techniques results in stable and long-term optimized behavior in both cases. Our experiments were performed on CAMNEP, an intrusion detection system based on analysis of NetFlow data, and were performed on the university network over one month.**

***Index Terms*—IDS, Adaptation, Game Theory, Regret minimization**

I. Introduction

Network Intrusion Detection/Protection Systems (IDS) are designed to identify and possibly block undesirable traffic detected on computer networks. Most of the current systems [1] are based on the signature matching paradigm: they inspect the content of the transferred data and look for the signatures of known attacks, in principal being similar to the anti-virus solutions. While being very effective against well-known threats, these systems are ineffective against new kinds of attacks: they are not able to combat threats that can not be defined by a simple signature, e.g. polymorphic malware, custom-written malware/attacks, malware command & control traffic or information exfiltration. Therefore, organizations that need an additional level of security typically protect themselves with the IDS systems based on the anomaly detection principle [2]. These systems do not try to identify specific attacks, but rather use past behavior of the network to predict its future behavior and report any significant differences between the prediction and the actual behavior.

The key parameters of the anomaly-based IDS are its *sensitivity* (detection rate or true positive rate) and the *false positive* rate. True Positives (TP) are the malicious events correctly detected by the system, while the False Positives (FP) are the false alarms raised by the system. Any practically usable IDS needs to keep the rate of false positives down, while maintaining the sensitivity. As the network conditions change over time, the IDS may need to change its behavior/configuration to adapt to the changing conditions. Formal approaches based on control theory are commonly used to formalize the adaptation process and to ensure that basic properties are satisfied in wide range of naturally occurring environments [3].

However, the control-based approach to reconfiguration is not without risks in when we face intelligent adversaries. The most sophisticated attackers are becoming aware of the anomaly-detection-based IDS and might be able to use specific techniques to circumvent them by inferring their internal state and modifying their behavior accordingly. They can also behave strategically in order to influence the internal state of the IDS towards lower sensitivity [4], [5], [6] or towards higher rate of false positives, both of which would make the system unusable. The existence of unsecured control mechanism increases the potential for the attack, and most formalisms based on control theory do not address adversarial manipulation. To address this threat, the adaptation mechanisms needs to be based on game-theoretical principles. The game theoretical methods are designed to carefully weight the optimality and predictability of system behavior [7], [8], [9] and to guarantee system properties against strategically acting attackers.

In our work, we consider yet another aspect of this problem: the actual feasibility and practicality of such strategic adaptation in an intrusion detection system, distributed both functionally and geographically. We argue that given the current level of de-facto standardization in the domain, it is reasonably simple to integrate an IDS from the components provided by multiple vendors, as the data transfer formats are typically robust and based on open standards or open-source implementations. On the other hand, the *adaptation communication/collaboration standards are virtually non-existant* and could make the strategic adaptation of such composed systems problematic.

II. Regret and Regret Minimization

Regret minimization is an algorithm that attempts to achieve long-term optimality in a sequence of identical games. It does not require any knowledge of other players' utility functions or the communication between players (players are denoted as P), but is based on independent loss minimization performed

by the individual players given the payoffs received for their past moves.

The regret, or more specifically the *external regret*, is defined [10], [11] as an ex-post evaluated *loss of utility* due to the suboptimal strategy selection - thus the term regret. Due to the properties of our problem as imposed by the design of our system, we assume *full information model*, i.e. we assume that player P has access to all payoff values of all strategies x from the set X in any time step in the past[1]. It shall be noted that the regret minimization method *does not* require the players to know the utility function and/or goals of the opponents – the strategy played by the agents is based only on the feedback that they receive.

We define l^t as the loss related to the use of one defender's strategy in the time step t. Note that the strategy is not explicitly denoted[2] in the notation. Then, we define the loss aggregated over time as:

$$L^T = \sum_{t=T-N}^{T} l^t \tag{1}$$

and the regret as follows:

$$R^T = L^T - \min(L^T) \tag{2}$$

This represents the difference of (hypothetical) payoff encountered upon (hypothetical) playing of this strategy (or using a particular strategy selection algorithm) and the best payoff available over the set of player's strategies X.

The regret value is then used in a *polynomial weights regret minimization* (PWM) algorithm [12]. In the PWM algorithm, the player P selects the strategy x with a probability that is inversely proportional to the regret. Formally, we define weight w^t assigned to each strategy x at time t and p^t, the corresponding probability of playing the strategy x. Initially at time 1, all the strategies x from the set X receive the identical weights and probabilities:

$$w^1 = 1 \textit{ and } p^1 = \frac{1}{|X|}, \forall x \in X \tag{3}$$

With increasing time, more and more games are played and the values w^t and p^t assigned to individual strategies are determined as follows (with $\eta = 0.2$, as the convergence proof [10] requires $\eta < 0.5$):

$$w^t = w^{t-1}(1 - \eta L^{t-1}) \tag{4}$$

$$W^t = \sum_{x \in X} w^t \tag{5}$$

$$p^t = \frac{w^t}{W^t} \tag{6}$$

Amortized loss for dynamic environment. The loss/regret function used in our experiments includes time-amortized loss accumulated over the last N time periods, rather than over the totality of history. The Eq. 1 is thus modified as follows:

$$L^T = \omega L^{T-1} + (1 - \omega) l^T \tag{7}$$

The use of Eq. 7 with $\omega = 0.5$ instead of simple loss accumulation accounts for the fact that real-world players can join and leave games dynamically, and that the characteristics of the game can change dynamically. This is a generalization of traditional regret definition, but it is consistent with previous work, and key properties of the PWM method still hold as shown in [11].

A. Properties

Regret minimization is a robust method that yields predictable results in a wide range of games. In a static environment, it can be shown that the use of the proper regret minimization algorithms (like the one described above) bounds the maximal loss achieved using external regret minimization by the term $O(\sqrt{T \log |X|})$ relative to the best loss achievable [10]. This is a general result that can be applied outside of game theoretic frameworks.

In the specific case of two-player, zero-sum games, the use of regret minimization will make the player's payoffs converge towards the value of the game, with the speed of convergence bounded by the term above. This result can be generalized for a far broader class of games. Hart and Mas-Colell show that if *all* players play regret minimization in a sequence of static games, the joint distribution of play converges [13], [11] to the set of *correlated equilibria* [12], [14] of the stage game. One of the important corollaries is that the probability of strategy switching decreases as well, making the players reach increasingly longer sequences of constant strategy play.

The correlated equilibrium is an extension of the well-known Nash equilibrium (i.e. stable point in the strategy space where none of the players benefits from unilateral deviation) by assuming that the players can either communicate, or can observe a common variable(s), or share a history of gameplay. All these specific examples are special cases of a *correlating device* [10]. Such a device produces a set of (correlated) signals, one for each player, which use for strategy selection in the game. It can be shown that when the signals are fully correlated (e.g. when all players share a single public signal), the correlated equilibria set equals the convex hull of Nash equilibria.

On the other hand, convergence to the smaller set of Nash equilibria is possible, but is guaranteed only in very specific types of game, and not guaranteed at all in the general case [15]. In particular, in the two player, zero sum game example mentioned above, the game converges, but counter-examples of non-convergent games can be found even for simple three player games, such as the Shapley game [16]. In non-zero sum games with more than two players, the regret-minimizing algorithm provides robust results when other approaches may fail.

The results of [17] suggest that regret minimization is also robust in zero-sum finite extensive-form games with perfect

[1]This contrasts with a *partial information model*, where the players only have access to the results of strategies actually played in the past.

[2]Some authors use index i to emphasize the loss l_i^t relevance to a particular strategy x_i. In our case, we omit this index to make the notation lighter in context of multi-player game as it will be introduced in Section III.

recall when applied across independent information sets in the game. The regret (counterfactual regret) is measured and minimized on information sets in the game and the authors show that minimization of the counterfactual regret in individual game stages bounds the overall regret of the global game – albeit only in a very specific class of games. This is an extremely important property, as it hints that at least some of the regret minimization properties may hold when we split one of the players into several partial players, each of them selecting a fraction of the original player's strategy.

It is also important to note that we do not oblige all players to use regret minimization (even if we do so in the experiments in Section V). Regret minimization also performs robustly against the rational players who simply play their own equilibria strategies obtained by other methods. When the opponents play stable non-equilibria strategies, regret minimization is likely to benefit from the sub-optimality.

The question of convergence is further complicated in the dynamic environment. The algorithm's convergence speed becomes critical, as it needs to converge within the time interval when the environment (which defines the structure of the game) is relatively stable, making the set of correlated equilibria also stable. This also implies that the value of the past history decreases with increasing time difference, making us adopt the amortized loss function as defined by Eq. 7 to find the balance between the robustness and speed of convergence.

III. Formal Problem Statement

We assume that one distributed intrusion detection system, which can be decomposed into autonomously acting agents, plays against one opponent. The **Defender** system P_D consists of two layers P_D^1, P_D^2, each layer consists of one or more agents that implement one functionality/activity. Defender's strategies x_D are drawn from the set X_D and can be decomposed into distinct individual layer strategies: x_D^1, x_D^2 from the mutually orthogonal subspaces X_D^1 and X_D^2 of the space X_D. We have implemented the **attacker** P_A (and P_A^1,P_A^2) as a *model of the attacker* implemented within the system [18]. Typically, this also consists of one or more agents that realistically represent the goals, utility functions and actions use by the real attackers. This is by no means necessary for the regret minimization approach to work. We have opted for this method as it is easier to determine the Nash equilibria and correlated equilibria in the game. The details of attacker modeling will be discussed in Section IV. Attacker's strategies x_A (same in both game variants below) are drawn from the set X_A and correspond to the realistic attack actions that may harm the protected network. Each strategy corresponds to one individual attack technique available to the attacker. For simplicity, we assume that attacker's action spaces are identical (X_A) in both layers when the game is played in the distributed form.

Two game variants are defined as follows:

- ***Global game:*** In this traditional formulation (Fig. 1) used as a benchmark, we build and solve a two player game of the attacker P_A against the defender P_D. Defender selects full strategies x_D for both layers simultaneously,

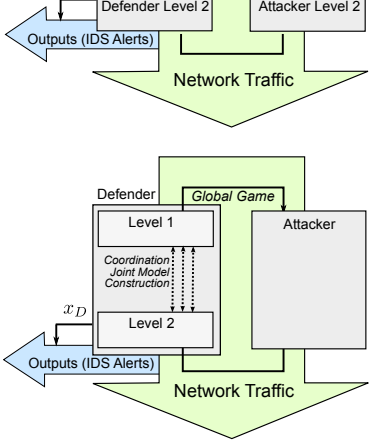


Fig. 1. Problem formulated as a global game.

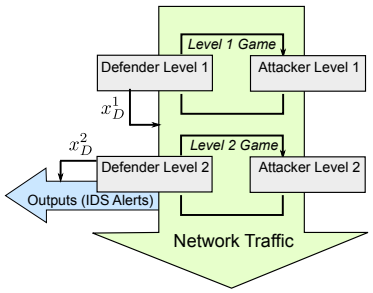


Fig. 2. Problem formulated as two games correlated through the effect on the network traffic and detection results. Note that both layers apply their strategies independently and that the strategy applied by the first layer influences the inputs of the second layer.

and has full access to all relevant system and environment state variables to build and update the global utility function u_D. In this setup, the defender is able to directly identify global optima in the global strategy space. On the other hand, finding the equilibria of the global game and reasoning about the selection of the optimal strategy becomes increasingly computationally difficult as the dimension of the strategy space grows. The game is formulated as follows:

$$G = (\{P_A, P_D\}, \{X_A, X_D\}, \{u_A, u_D\}) \quad (8)$$

$$u_A : x_A \times x_D \rightarrow \mathbb{R} \quad (9)$$

$$u_D : x_A \times x_D \rightarrow \mathbb{R} \quad (10)$$

The utility functions of the attacker u_A and the defender u_D are defined as functions of the global strategies.

- ***Distributed game:*** In our formulation of the game (Fig. 2), each of the layers P_D^1 and P_D^2 (and P_A^1 and P_A^2) plays as an independent player. Therefore, we have a four player game, with the players $P_A^1, P_A^2, P_D^1, P_D^2$, where the couples of the players **should** be ideally optimizing a shared utility function. In practice, though, the utility functions (on the defender's side) differ, as the internal state of the players P_D^1 and P_D^2 is mutually inaccessible (for integration reasons), meaning that their dynamically updated utility functions can only rarely be identical. This

defines the distributed version of the game as follows:

$$G = (\{P_A^1, P_A^2, P_D^1, P_D^2\}, \{X_A, X_D^1, X_D^2\}, \{u_A^1, u_A^2, u_D^1, u_D^2\}) \quad (11)$$

$$u_A^1 : x_A^1 \times x_D^1 \to \mathbb{R} \quad (12)$$

$$u_D^1 : x_A^1 \times x_D^1 \to \mathbb{R} \quad (13)$$

$$u_A^2 : x_A^1 \times x_A^2 \times x_D^1 \times x_D^2 \to \mathbb{R} \quad (14)$$

$$u_D^2 : x_A^1 \times x_A^2 \times x_D^1 \times x_D^2 \to \mathbb{R} \quad (15)$$

For higher generality, we break down both players and also **both** player's utility functions into u_A^1 and u_A^2 for the attacker and u_D^1 and u_D^2 for the defender. In practice, most attackers will just use a single strategy and utility function in both levels of the game, resulting in $u_A^1 = u_A^2$ and $x_A^1 = x_A^2$, and the game will degenerate into a three player game[3]. Note the distinction between the two layer's utility functions, where the first layer's results are independent of the second layer's actions, while the results of the second layer depend on both. Effectively, the utility function u_D^2 would typically be the same as u_D, with the notable difference that only the action x_D^2 is under the control of the player P_D^2, while the action x_D^1 is selected independently by the player P_D^1, who optimizes its own utility function u_D^1. We do not require the players P_D^1 and P_D^2 to know each other's utility functions, but their consistence with u_D determines whether the distributed game results in the same equilibria as the global game.

The notion of independent optimization in the distributed game where only one of the layers depends on the actions of the other, and is not explicitly informed about the action played by the first layer, makes the game consistent with the formalization based on extensive-form games introduced in [17] and discussed in Sec. II-A. As the defender's second layer is not informed about the strategy x_D^1 played by the first layer, it performs the selection of x_D^2 in the same information set regardless of the x_D^1. Likewise, the first defender player is typically not able to infer the strategies played, payoffs received or the internal variable states of the second level player.

Both formulations will be solved by the regret minimization algorithm introduced in Section II, in one case by two players, in the other by four players playing independently. We are facing a dynamic optimization problem, where the environmental conditions imposed by the external environment can change rapidly, making the game similar to a sequence of static games with unpredictable length. In the next section, we will introduce a specific instantiation of the above model on a specific intrusion detection system.

[3]The three player formulation would be:

$$G = (\{P_A, P_D^1, P_D^2\}, \{X_A, X_D^1, X_D^2\}, \{u_A, u_D^1, u_D^2\}) \quad (16)$$

$$u_A : x_A \times x_D^1 \times x_D^2 \to \mathbb{R} \quad (17)$$

$$u_D^1 : x_A \times x_D^1 \to \mathbb{R} \quad (18)$$

$$u_D^2 : x_A \times x_D^1 \times x_D^2 \to \mathbb{R} \quad (19)$$

Graphically, this would correspond to the attacker from Fig 1 playing against the defenders from Fig. 2

IV. Experimental System

The experiments were performed inside the CAMNEP [19] Intrusion Detection System which was de-coupled into two layers to allow the evaluation of the above-described principles. The first layer is the **preprocessing layer**, where the NetFlow/IPFIX [20] data from the network is received, preprocessed and possibly sampled using a dynamically selected **sampling strategy**. The preprocessing layer is followed by the **detection layer** where a set of anomaly detection methods performs a collective analysis of the received data using statistical and information models of network traffic. The detection layer of CAMNEP has been designed to adapt to the network situation and to select the optimal **aggregation strategy** – the mix of weights to assign to the outputs of the anomaly detection methods – so that the system can dynamically find the best combination of opinions for the given environment. Selection of both the sampling algorithm, rate, and the appropriate aggregation strategy (that are obviously linked) are crucial for system performance.

Characterization of the immediate system performance is a difficult problem, as manual feedback is nearly impossible due to the sheer volume, and rate of input data, and general unavailability of any ground truth about their nature. Therefore, we use a challenge insertion mechanism that modifies the input data of the system by inserting a small number of classified events from the past, belonging to both legitimate traffic and various attack classes. These **challenges** [18] are processed alongside of the real input data, used for system component/strategy characterization/evaluation, and then removed from the output data produced by the system.

The game as described below is actually played between **two opponents inside the system**. One of them, the defender P_D, uses the challenge processing results to select the best sampling strategy x_D^1 and aggregation strategy x_D^2 (whose combination in the global game is denoted x_D). It then plays against the attacker's model P_A that determines and exploits the system vulnerabilities in the same manner as the worst-case real attacker would by playing its attack strategies represented by challenges: x_A^1, x_A^2 or the global strategy x_A. The dynamic optimization achieved by this method is crucial, as the system constantly reconfigures itself to counter the worst case attacker (with full access to its internal state) on the background of the current network traffic.

Specifically, the lower layer has to select one of five predefined sampling methods, each with two different ratio settings. The detection layer has to select a single aggregation function from a static finite set.

The system operates sequentially, processing one batch of data every five minutes. Each data set defines one game in the sequence. When the challenges are inserted, all players use the results of their processing to determine the regret value for each available strategy and then commit to the strategy selected by the algorithm presented in Sec. II. The

final strategies of the defender (or defenders) are then used to process real network data from the current batch.

Defender (CAMNEP) has to satisfy safety demands, it tries to lower the probability of undetected attacks, and at the same time optimizes the usage of computational resources. Conversely, the attacker tries to attack the system while avoiding detection. Thus the attacker has to find the best attack type and size to fit under the detection threshold by exploiting the flaws either in the sampling or aggregation strategy.

A. Global Game

In the global game, the defender performs global optimization and identifies the best strategy x_D from the set X_D, effectively looking for the best combination of input data sampling and the anomaly detector's aggregation scheme.

The utility function of the defender in the global game has three principal components: the first term deals with successfully detected attacks, the second term represents the loss associated with undetected attacks, and the third term describes the overhead of the monitoring, which consists of the false positive costs and the fixed cost of monitoring.

Individual utility functions of the global game for both players (adapted from [7] and made more realistic) are defined as follows. Defender's utility is:

$$u_D(x_D, x_A) = \alpha(D_D(x_D) - C_{TP}) + (1-\alpha)\gamma P_D(x_A) - \beta V C_{FP} - C_M \quad (20)$$

where α denotes the probability that a particular attack strategy x_A is detected when the defender selects the defense strategy x_D; $D_D(x_A)$ denotes the defender's payoff for attack detection; C_{TP} denotes the (average) cost of processing of each successfully detected incident (true positive) for the defender; γ denotes the probability of attack success; $P_D(x_A)$ denotes the defender's payoff/loss on attack success; β denotes the probability that a given detection strategy, combined with current system state and background traffic, will result in a false positive; V denotes the background traffic volume that is used to estimate the number of false positives in combination with the parameter β; C_{FP} denotes the average cost of a false alarm for the defender, used in conjunction with β and V to estimate the false positive cost; and C_M denotes the fixed cost of monitoring infrastructure, independent on attack or traffic intensity.

Attacker's utility can be described as:

$$u_A(x_D, x_A) = \alpha D_A(x_A) + (1-\alpha)\gamma P_A(x_A) - C_A(x_A) \quad (21)$$

where α and γ are the same as described above; $D_A(x_A)$ denotes the attacker's payoff/loss on detection; $P_A(x_A)$ denotes the expected utility the attacker receives upon successful realization of given attack action from the attack class corresponding to strategy x_A; $C_A(x_A)$ denotes the cost of the attack performance on the part of attacker.

B. Distributed Game

As we have defined in Sec. III, the optimization problem in the distributed game is separated into two different games solved independently. Thus, we have to define four different utility functions for four independent players P_A^1, P_D^1 and P_A^2, P_D^2.

On the first layer, the player P_D^1, i.e. the defender, optimizes the selection of the best sampling method which preserves as much information necessary for detection as possible and at the same time optimizes the volume of traffic. These contradictory requirements can be summarized into following equation:

$$u_D^1(x_D^1, x_A^1) = \xi_D(1 - \frac{|\Phi_s(x_D^1)|}{|\Phi|}) + \vartheta_D \frac{|I_s(x_D^1)|}{|I|} + \mu_D |\{c \in C || c_s(x_D^1)| > \varepsilon\}| \quad (22)$$

where $|\Phi_s(x_D^1)|$ is the number of flows in the dataset after the sampling using the strategy x_D^1; $|\Phi|$ represents the number of flows in the whole dataset before sampling; $|I_s(x_D^1)|$ is the number of distinct IP addresses in the sampled dataset; $|I|$ is a number of distinct IP addresses in the unsampled dataset; C is the set of challenges inserted into the traffic, as described in Section IV and $|c_s(x_D^1)|$ represents the number of flows from inserted challenge c after sampling, using method x_D^1. The first additive term emphasizes the need to sample as much traffic as possible, the second, emphasizes the need to preserve as much diversity as possible, using the source IP as the most important feature, and the third term represents the need to keep enough information from each challenge (here used as a representative of a typical incident) for further analysis. All three terms are contradictory, and $\xi_D = 1$, $\vartheta_D = 2$ and $\mu_D = \frac{1}{2}$ are constants assigning the relative importance to the different parts of the utility.

Attacker's utility function can be summarized as follows:

$$u_A^1(x_D^1, x_A^1) = \frac{1}{|C|} \sum_{c \in C} (1 - \frac{|c_s(x_D^1)|}{|c|}) \quad (23)$$

where C is the set of all challenges inserted in the current dataset, term $|c|$ represents the volume of the challenge c, i.e. the number of flows and $|c_s(x_D^1)|$ again represents the number of sampled flows from challenge c obtained using the sampling strategy x_D^1. Thus, from Eq. 23, it can be seen that the goal of the attacker in the first level of the game is to perform an attack with as high volume of traffic as possible, thus increasing the speed of probing or brute force operations, which are sampled as little as possible.

Utility functions in the detection layer are nearly as identical in the global game as they are for both attacker and defender. This is logical, as their values determine the same output with the only difference being formal. This is due to the fact that the first level sampling strategy is imposed by the first layer, rather than optimized together with the aggregation (or attack) strategy.

The second layer of neither the attacker, nor the defender gets information regarding the strategies used in the first layer,

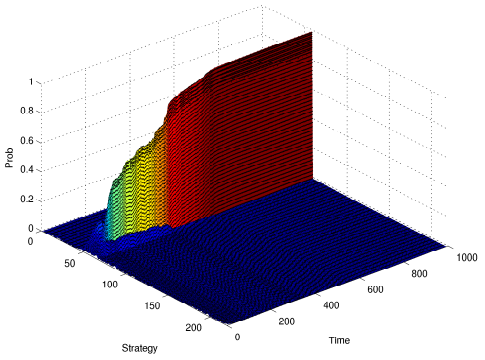


Fig. 3. Convergence of regret minimization in the global game. Note far larger strategy space which is finally reduced into the single strategy.

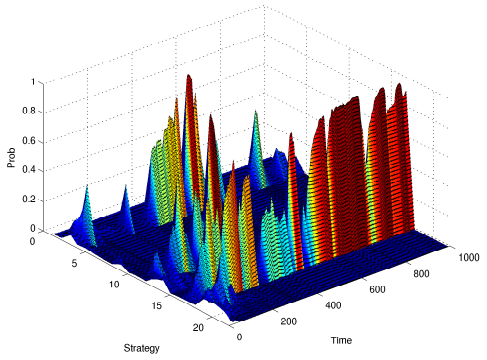


Fig. 4. Selection probabilities obtained by the PWM algorithm in the defender's detection game.

and the only implicit information passed between the layers is the shape of the data after the application of the sampling strategy.

V. Experimental Evaluation

The experiments presented bellow were performed on the CAMNEP system deployed on the university network and processing the data from single campus building – 200-300 users, with approximate bandwidth in hundreds of Mbits/sec and about 30-50 thousand flows in each 5-minute long dataset. We have used 1000 datasets in a series, capturing roughly four days of traffic and in one case, we have verified the properties on about 8 000 datasets, extending the runtime to 28 days.

The first issue to validate is the convergence of the global game towards the set of correlated equilibria (delimited by the Nash equilibria). Even before that, we need to determine whether the global game converges from the defender's perspective, i.e. whether the strategies selected by the regret minimization algorithm are stable. In Fig 3, we can see that the global player can chose from 220 different strategies, defined as a Cartesian product of 10 sampling strategies and 22 detection strategies. We can clearly see that the behavior is stable. During the first 100 datasets (about 8 hours), the system tries several strategies before slowly converging to a single pure strategy which becomes dominant after about 1 day of uptime.

We can also see that the sampling game (Fig. 5) behaves similarly, although on a much smaller support. At the beginning, it quickly converges to the sequence of mixed strategies with two components, but the composition of couples evolves over time, before it stabilizes after about 200 time periods. However, the fact that the lower layer settled on a 50:50 mix of two strategies causes major problem for the the second detection layer, as it can only guess which of these two pure strategies is currently being played. In Fig. 4, we can observe the effects on the detection layer. At the beginning (first 50 datasets), there are 4 strategies with significant values of p^t, i.e. with non-negligible probabilities of being played. Then, the system settles on a mixed equilibria of two strategies (17 and 12), before settling on a combination of the long-term

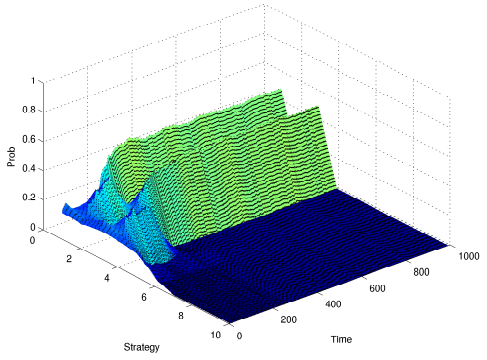


Fig. 5. Selection probabilities of individual strategies in the defender's sampling game. Note the progressive disappearance of most strategies, rapid convergence at first to the strategies 2 and 4, and finally to 1 and 3.

dominant strategy number 17 occasionally complemented by the strategy number 4.

The dominant detection game strategy picked in both the global and distributed game is the same – the detection part of the strategy selected in the global game is strategy 17 from Fig. 4. On the other hand, the sampling strategies selected in both games overlap, but are not identical. The global game selects the optimal choice, while the distributed version of the sampling game alternates this selection with the second, very similar strategy.

Above, we have confirmed that both games converge and rapidly achieve robust behavior. However, we still need to assess whether the regret minimization algorithm is able to identify the Nash equilibria, as this is not guaranteed in a dynamic system. In Fig. 6, we can see that the payoffs realized by the global game in both the sampling stage and the detection stage quickly converge towards the region of correlated equilibria. These sets of correlated equilibria are not shown for clarity, and we only depict the pure Nash equilibria that delimit them. This is not a problem, as we can see that the convergence is quick and robust in both dimensions. We can therefore conclude that in our particular case, the system even outperforms the theoretical guarantees and the strategies selected by the regret minimization coincide with the Nash

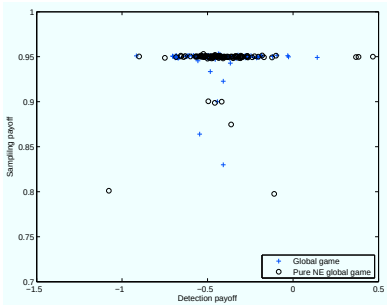


Fig. 6. Payoffs for the defender's strategies in the global game plotted against the positions of the pure Nash equilibria. X axis shows the detection payoff, while the Y axis shows the sampling payoff for each strategy selected or the Nash equilibria. Only randomly selected (1 out of 10) results shown for clarity.

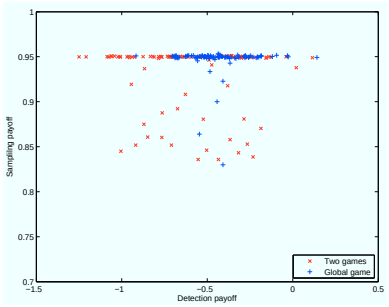


Fig. 8. Payoffs of the defender's strategies for both the global game (from Fig. 6) and the distributed game (from Fig. 7), plotted against each other. Only randomly selected (1 out of 10) results from the first 500 datasets shown for clarity. Note the lower dispersion and higher payoff in the detection dimension for the global game results.

equilibria in the long run.

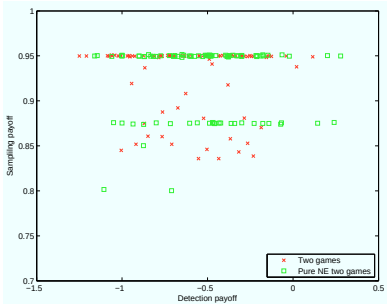


Fig. 7. Payoffs for the defender's strategies in the distributed game plotted against the positions of the pure Nash equilibria. X axis shows the detection payoff, while the Y axis shows the sampling payoff for each strategy selected or the Nash equilibria. Only randomly selected (1 out of 10) results shown for clarity. Note larger dispersion and the effect of mixed strategy selection in the sampling game - two payoff levels are clearly visible for both the equilibria and the game results.

The convergence of the distributed game is slower. Fig. 7 shows that the detection game payoff also converges into the region of Nash/correlated equilibria, but that this region is significantly larger (wider) than in the global game (Fig. 6). This is due to the fact that the regret minimization in the sampling game results in mixed equilibria, as discussed above. In Fig. 7, we can see the smaller payoffs corresponding to the selection of sampling strategies other than the strategy number 17. Use of the other strategies (mainly strategy 4) results in sampling payoffs lower than the standard 0.95. Fig. 8 shows the combination of the above figures. We can see that the global game payoffs in the detection dimension are on average higher than the corresponding payoffs for the distributed game, a point that we will investigate further.

VI. Related Work

The problem of reconfiguration and parametrization (both runtime and offline) of intrusion detection systems has been addressed from several perspectives, even if the direct use of game-theoretic principles is relatively rare and most contributors present the solutions applied for offline use [21], [22], [7], [8], when they identify the environment parameters static system and adapt the IDS to this static viewpoint. Roy et al. [9] present an overview of game-theoretical models of the network intrusion detection problem.

The initial work of Alpcan [21] analyzes the IDS game as a sequence of interactions between strategically reasoning opponents and a network of IDS sensors, in the format of two player, single act finite game with dynamic information. In [22], Alpcan and Basar extend [21]. Their formalism, based on a combination of Markov games and Q-learning, links the agent's performance as a detector/learner to its game performance by representing the imperfect information. The utility function that we use is based on the work of Chen [7], but we have included supplementary terms that represent real world concerns, and make the game decidedly a non-zero sum game.

In [23], Al-Nashif et al. proposed a multi-level IDS. In their system, they use various intrusion detection techniques on different levels, such as signature-based intrusion detection and stateful protocol analysis. In order to fuse the different levels, they compute a linear combination of their outputs. If the different levels disagree too much on their classification, they are separately trained by means of supervised learning. We follow another approach where the different agents are cross-linked to optimize their classifications. Then, we eventually choose the best-performing agent that is selected according to the explicit detection priorities specified by the threat model. An alternative approach to multi-model intrusion detection is based on the use of ensemble classification approaches [24]. However, these techniques require a pre-classified training data set and do not dynamically adapt system to the changing conditions.

VII. Conclusions

This paper contributes to the development of robust and survivable distributed intrusion detection systems (and also other systems where the same abstract model can be fit) by introducing a collective adaptation paradigm which is not based on explicit communication, but on the individual adaptation of components working on the shared data. From

a research perspective, our work is based on results regarding the convergence of regret minimization approaches to the set of correlated equilibria [11][10] and a more restricted result regarding the bounds of regret minimization in information sets of extended games [17]. Both theoretical results suggest that explicit communication and collaboration only brings limited benefit, at the expense of far worse system flexibility.

Our work quantifies this benefit and analyzes its impact on an actual system which can be expected to run in the adversarial environment. The experimental results show that:

- both the global and distributed implementation of regret minimization do converge towards the correlated equilibria/Nash equilibria and the most-frequently played strategies are identical for both approaches,
- the distributed game presents more variable behavior, i.e. it tends to converge towards a mixed equilibria, which is less predictable by the opponent, but,
- the global version of the game outperforms the distributed version from the utility maximization point of view, due to the price of non-synchronization.

Our results suggest that independently optimized components would perform very well under most circumstances where the slight sub-optimality would be compensated by the ease of integration, fewer maintenance problems and more open integration/reconfiguration options. Both the global approach and the local approach provide result in robust behavior which in some aspects exceeds the theoretical guarantees by actually converging into the Nash equilibria. Furthermore, the individual adaptation of components results in globally rich and nearly unpredictable behavior, which is by its nature resistant to modeling through adversarial machine learning techniques [6], [4]. Furthermore, the lack of coordination communication between agents and corresponding interfaces reduces the IDS attack surface visible to the sophisticated attacker and make the system more difficult to exploit.

ACKNOWLEDGMENT

This material is based upon work supported by the ITC-A of the US Army under Contract No. W911NF-10-1-0070. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ITC-A of the US Army. Also supported by Czech Ministry of Education grant AMVIS-AnomalyNET, MSMT ME10051 and MEB111008.

REFERENCES

[1] K. Scarfone and P. Mell, "Guide to intrusion detection and prevention systems (idps)," NIST, US Dept. of Commerce, Tech. Rep. 800-94, 2007.

[2] D. E. Denning, "An intrusion-detection model," *IEEE Transaction Software Engineering*, vol. 13, no. 2, pp. 222–232, 1987.

[3] J. L. Hellerstein, "Why feedback implementations fail: the importance of systematic testing," in *Proceedings of the Fifth International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, ser. FeBiD '10. New York, NY, USA: ACM, 2010, pp. 25–26. [Online]. Available: http://doi.acm.org/10.1145/1791204.1791209

[4] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?" in *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security*. New York, NY, USA: ACM, 2006, pp. 16–25.

[5] M. Barreno, P. L. Bartlett, F. J. Chi, A. D. Joseph, B. Nelson, B. I. Rubinstein, U. Saini, and J. D. Tygar, "Open problems in the security of learning," *Conference on Computer and Communications Security*, pp. 19–26, 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1456377.1456382

[6] N. N. Dalvi, P. Domingos, Mausam, S. K. Sanghai, and D. Verma, "Adversarial classification," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*. ACM, 2004, pp. 99–108.

[7] L. Chen and J. Leneutre, "A game theoretical framework on intrusion detection in heterogeneous networks," *Information Forensics and Security, IEEE Transactions on*, vol. 4, no. 2, pp. 165–178, June 2009.

[8] G. Wagener, R. State, A. Dulaunoy, and T. Engel, "Self adaptive high interaction honeypots driven by game theory," in *SSS*, 2009, pp. 741–755.

[9] S. Roy, C. Ellis, S. Shiva, D. Dasgupta, V. Shandilya, and Q. Wu, "A survey of game theory as applied to network security," in *HICSS*. IEEE Computer Society, 2010, pp. 1–10.

[10] A. Blum and Y. Mansour, "learning, regret minimization and equilibria," in *Algorithmic Game Theory*, N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, Eds. Cambridge University Press, 2007, ch. 4, pp. 79–101.

[11] S. Hart, "Adaptive Heuristics," *Econometrica*, vol. 73, no. 5, pp. 1401–1430, Sep. 2005. [Online]. Available: http://www.blackwell-synergy.com/doi/abs/10.1111/j.1468-0262.2005.00625.x

[12] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic Game Theory*. New York, NY, USA: Cambridge University Press, 2007.

[13] S. Hart and A. Mas-Colell, "A simple adaptive procedure leading to correlated equilibrium," *Econometrica*, vol. 68, no. 5, pp. 1127–1150, September 2000.

[14] R. Aumann, "Correlated equilibrium as an expression of Bayesian rationality," *Econometrica: Journal of the Econometric Society*, 1987. [Online]. Available: http://www.jstor.org/stable/1911154

[15] S. Hart, "Nash equilibrium and dynamics," Center for Rationality and Interactive Decision Theory, Hebrew University, Jerusalem, Discussion Paper Series dp490, Sep. 2008.

[16] J. Shamma and G. Arslan, "Dynamic fictitious play, dynamic gradient play, and distributed convergence to Nash equilibria," *IEEE Transactions on Automatic Control*, vol. 50, no. 3, pp. 312–327, Mar. 2005. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1406126

[17] M. Zinkevich, M. Johanson, M. H. Bowling, and C. Piccione, "Regret minimization in games with incomplete information," in *NIPS*, J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, Eds. MIT Press, 2007.

[18] M. Rehák, E. Staab, V. Fusenig, M. Pechoucek, M. Grill, J. Stiborek, K. Bartos, and T. Engel, "Runtime monitoring and dynamic reconfiguration for intrusion detection systems," in *Recent Advances in Intrusion Detection, 12th International Symposium, RAID 2009, Saint-Malo, France, September 23-25, 2009. Proceedings*, E. Kirda, S. Jha, and D. Balzarotti, Eds., 2009, pp. 61–80.

[19] M. Rehák, M. Pechoucek, M. Grill, J. Stiborek, K. Bartoš, and P. Celeda, "Adaptive multiagent system for network traffic monitoring," *IEEE Intelligent Systems*, vol. 24, no. 3, pp. 16–25, 2009.

[20] B. Claise, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information," RFC 5101 (Proposed Standard), Internet Engineering Task Force, Jan. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5101.txt

[21] T. Alpcan and T. Başar, "A game theoretic approach to decision and analysis in network intrusion detection," in *Proceedings of the 42nd IEEE Conference on Decision and Control*, Maui, HI, December 2003, pp. 2595–2600. [Online]. Available: papers/cdc03_alpcan_WeP03-1.pdf

[22] ——, "An intrusion detection game with limited observations," in *12th Int. Symp. on Dynamic Games and Applications*, Sophia Antipolis, France, July 2006. [Online]. Available: papers/isdg06.pdf

[23] Y. Al-Nashif, A. A. Kumar, S. Hariri, Y. Luo, F. Szidarovsky, and G. Qu, "Multi-level intrusion detection system (ml-ids)," in *ICAC '08: Proceedings of the 2008 International Conference on Autonomic Computing*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 131–140.

[24] G. Giacinto, R. Perdisci, M. D. Rio, and F. Roli, "Intrusion detection in computer networks by a modular ensemble of one-class classifiers," *Information Fusion*, vol. 9, no. 1, pp. 69–82, 2008.

Game Theoretical Adaptation Model for Intrusion Detection System*

(Extended Abstract)

Martin Rehak†‡, Michal Pechoucek†‡, Martin Grill†, Jan Stiborek†, Karel Bartos†
† Department of Cybernetics, Czech Technical University in Prague, Czech Republic
‡ Cognitive Security s.r.o., Prague, Czech Republic
martin.rehak@agents.felk.cvut.cz

ABSTRACT

We present a self-adaptation mechanism for Network Intrusion Detection System which uses a game-theoretical mechanism to increase system robustness against targeted attacks on IDS adaptation. We model the adaptation process as a strategy selection in sequence of single stage, two player games. The key innovation of our approach is a secure runtime game definition and numerical solution and real-time use of game solutions for dynamic system reconfiguration. Our approach is suited for realistic environments where we typically lack any ground truth information regarding traffic legitimacy/maliciousness and where the significant portion of system inputs may be shaped by the attacker in order to render the system ineffective. Therefore, we rely on the concept of challenge insertion: we inject a small sample of simulated attacks into the unknown traffic and use the system response to these attacks to define the game structure and utility functions. This approach is also advantageous from the security perspective, as the manipulation of the adaptive process by the attacker is far more difficult. Our experimental results suggest that the use of game-theoretical mechanism comes with little or no penalty when compared to traditional self-adaptation methods.

Categories and Subject Descriptors

C.2.0 [**COMPUTER-COMMUNICATION NETWORKS**]: General—*Security and protection*

General Terms

Algorithms, Security

Keywords

adaptation, game theory, security, intrusion detection

*This material is based upon work supported by the ITC-A of the US Army under Contract No. W911NF-10-1-0070. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ITC-A of the US Army. Effort is also sponsored by the Air Force Office of Scientific Research, Air Force Material Command, USAF, under grant number FA8655-10-1-3016. Also supported by Czech Ministry of Education grants 6840770038 and ME10051.

Cite as: Game Theoretical Adaptation Model for Intrusion Detection System (Extended Abstract), M. Rehak, M. Pechoucek, M. Grill, et al., *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems – Innovative Applications Track (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. XXX-XXX.

1. INTRODUCTION

In this paper, we use the game-theoretical models to improve the security of the adaptation process within a distributed, agent-based Intrusion Detection System (IDS). The high-level self-adaptation method that we develop our approach on [2] has been designed for the intrusion detection systems based on the anomaly detection paradigm: these systems observe the past behavior of the monitored network/hosts, predict their future behavior using statistical and other models and identify the behavior diverging from the prediction as anomalous. Adaptation, self-management and self-optimization techniques that are used inside an IDS can significantly improve their performance [2] (i.e. reduce the number of false alarms) in a highly dynamic environment, but are also a potential target for an informed and sophisticated attacker. When the adaptation techniques are deployed improperly, they can alow the attacker to reduce the system performance against one or more critical attacks. This paper presents a game theoretical model of adaptation processes inside an autonomic, self-optimizing IDS, presents an architecture integrating the process with an existing IDS.

We present an **architecture** that integrates the abstract game model into an IDS with self-monitoring capability, in order to simulate the worst case, optimally informed attacker and to optimize the system behavior against such attacker. Such (hypothetical) attacker with full access to system parameters could dynamically identify the best strategy to play against the system. Optimizing the detection performance against the worst case attacker protects the system from more realistic attacks based on long-term probing and adversarial machine learning approaches referenced above.

2. GAME MODEL

We conceptualize the relationship between the attacker and the defender as a *sequence* of *single stage, two player, non-zero sum games*, where the attack/defence actions of both players correspond to strategies in the game-theoretical model of their interaction and the environment evolves between the game. The game model (and utility functions in particular) are based on [1], with additional inputs from the network administrators and actual IDS users. The game model integrates the preferences and strategies of two players (attacker and defender). Their strategy sets are defined as a selection of IDS configurations for the defender and the selection of a particular attack type (e.g. buffer overflow, password brute-force, scan...) for the attacker. The main difference of the utility functions from [1] is the relaxation of the requirement on the identical attacker gain/defender loss and the proportionality of associated costs (alarm processing, monitoring etc.) with the gain/loss value. This requirement was considered as too strong by the system administrators we have questioned.

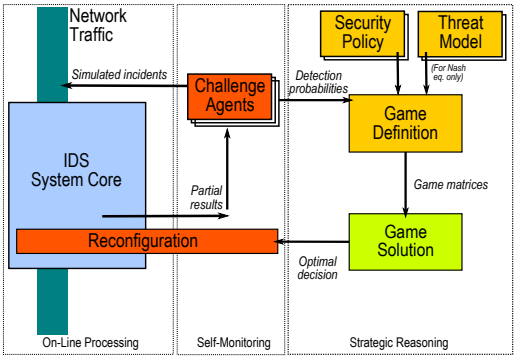


Figure 1: Indirect online variant of game/IDS integration.

The actual utility function values of both players depend principally on the sensitivity of the system using defender's strategies with respect to individual attacker's strategies ($\alpha_{i,j}$) , and the associated rate of false positives (β_i) for each configuration. $\alpha_{i,j}$ denotes the probability that the j-th attack strategy is detected by the IDS when the defender plays the i-th defence strategy and β_i denotes the probability that the i-th defender's strategy will result in a false alert. These parameters shape the utility functions of both players in each game stage. By our experience, these values wary widely with changing characteristics of the background traffic and need to be estimated dynamically for each given game in a sequence, as we will present below.

The gameplay is very simple in our case: both players simultaneously select their strategies from the set S and the combination of these strategies determines the payoffs to attacker and defender, as defined by their respective utility functions. The solution concepts used to solve/analyze the game are **Max-Min** and **Nash** equilibria. We play a sequence of games described above, each corresponding to one time interval. The individual games in the sequence are differentiated by the dynamically evolving parameters of player's utility functions. We consider the individual games to be independent and we don't carry over any information between them.

3. ARCHITECTURE

There are two existing approaches to integration of the game model with an IDS:

◇ *Off-line integration*, when the game is defined in design time, solved analytically, using *a priori* knowledge about expected impacts and success likelihood of the attacks, and the system parameters are fixed to resulting strategies according to game results. Game theory use ensures that the system parameters are set to force the adversary into the selection of less damaging (or more rational) strategies. It is sufficient for systems deployed in stable environments, but most IDS need to cope with dynamic environments, where the background traffic an other factors change frequently. In such environments, the static strategies perform poorly.

◇ *Direct on-line integration*, when the game uses presumed adversary actions in the observed network traffic to define the game is the opposite approach. The game is being defined by the actual actions of real-world attackers executed against the monitored system, elegantly solving the relevance problem. On the other hand, direct interaction between the adversary and the adaptation mechanism makes the system potentially vulnerable to attacks against the adaptation algorithms, creating a new attack surface. Motivated attacker can easily mislead the IDS by insertion of a sequence of attacks that are orthogonal to its actual plan to target its utility.

Our approach, named *indirect online integration* combines the above approaches and provides interesting security properties desirable for real-world deployment. The solution uses the concept of challenges [2] to mix a controlled sample of legitimate and adversarial behavior with actually observed network traffic and is a compromise between the above approaches (see Fig. 1). In this case, the real traffic background (including any possible attacks) is processed in conjunction with simulated hypothetical attacks within the system. We measure the system response to these challenges, drawn from the realistic attack classes, and use them to estimate the system response to the real-world samples from the same classes. In practice, we will define one class for each broadly defined attack/legitimate traffic type and measure the difference between the system response to legitimate traffic and to various classes of malicious traffic. The challenges are then mixed with the real traffic on IDS input and the system response to them is used as an input for game definition, measuring/estimating the current values of: $\alpha_{i,j}$ and β_i. The major advantage is higher robustness w.r.t strategic attacks on adaptation algorithms, and lower system configuration predictability by the adversary, as the simulation runs inside the system itself and its results can not be easily predicted by the attacker.

This approach offers the optimal mix of situation awareness and security against engineered inputs. In this case, we actually play against an abstract opponent model inside the system, and expect that the moves that are effective against this opponent will be as effective against the real attacks. The advantage of this approach is not only in its security, but also in better model characteristics in terms of strategy space coverage (unfrequent, but critical attacks are covered), robustness and relevance – the abstract game can represent the attacks and utility combinations that would be obvious only for insider attackers.

4. CONCLUSIONS

The experiments we have performed with a simplified (and modified) version of commercially available IDS solution clearly showed that the game theoretical models/solvers integrated into an adaptive IDS provide the results more than equivalent to the alternative direct optimization methods, as we have verified on inserted challenges and real-world attacks performed on the monitored network. These methods provide robust performance and reliably converge when using both max-Min or Nash equilibria. The additional benefits, such as increased robustness against an attacker with insider access, therefore build a strong case for their use by the industry. In particular, our results suggest that the max-min solution concept provides very consistent results, does not require an explicit model of opponent's utility function and is computationally trivial, making it an interesting first choice for future proof-of-concept implementations.

5. REFERENCES

[1] L. Chen and J. Leneutre. A game theoretical framework on intrusion detection in heterogeneous networks. *Information Forensics and Security, IEEE Transactions on*, 4(2):165–178, June 2009.

[2] M. Rehak, E. Staab, M. Pechoucek, J. Stiborek, M. Grill, and K. Bartos. Dynamic information source selection for intrusion detection systems. In K. S. Decker, J. S. Sichman, C. Sierra, and C. Castelfranchi, editors, *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '09)*, pages 1009–1016. IFAAMAS, May 2009.

Optimizing Flow Sampling for Network Anomaly Detection

Karel Bartos, Martin Rehak, Vojtech Krmicek

***Abstract*—Sampling techniques are widely employed in high-speed network traffic monitoring to allow the analysis of high traffic volumes with limited resources. Sampling has measurable negative impact on the accuracy of network anomaly detection methods. In our work, we build an integrated model which puts the sampling into the context of the anomaly detection used in the subsequent processing. Using this model, we show that it is possible to perform very efficient sampling with limited impact on traffic feature distributions, thus minimizing the decrease of anomaly detection efficiency. Specifically, we propose an adaptive, feature-aware statistical sampling technique and compare it both formally and empirically with other known sampling techniques – random flow sampling and selective sampling. We study the impact of these sampling techniques on particular anomaly detection methods used in a network behavior analysis system.**

I. Introduction

Sampling is frequently used by network monitoring and intrusion detection devices to handle the large volumes of traffic in a cost-effective manner. There are many sampling methods available (see their discussion in Section II), but these methods are mainly optimized to preserve the low-level parameters, such as traffic volume or number of packets. However, the use of sampled data for more advanced analysis, such as Network Behavior Analysis, is problematic [10], as the sampling severely harms the effectiveness of the anomaly detection and data analysis algorithms. These algorithms are based on pattern recognition and statistical traffic analysis, and the distortion of traffic features can significantly increase the error rate of these underlying methods by breaking their assumptions about traffic characteristics.

In our paper, we analyze flow-level sampling of NetFlow/ IPFIX information, and provide three principal contributions:

- A *formal model* integrating the sampling with the subsequent stages of processing, which allows us to analyze the impact of sampling on anomaly detectors used in the subsequent processing of information. (Section III)
- A concept of *late sampling*, which significantly improves the performance of sampling methods by interleaving them with early stages of anomaly detection. (Sec. IV-A)
- An adaptive sampling method, which optimizes the sampling algorithm behavior w.r.t. the anomaly detection algorithms used later in the processing. (Section IV-B)

The improvements, which are clearly outlined in Section V, come with a price. The techniques outlined in this paper are based on a violation of the traditional layering applied to traffic monitoring methods. The lower layers of network monitoring solutions, that actually perform the sampling, need to be aware of the traffic features used by the higher layer algorithms, and need to modify their behavior following any reconfiguration of the upper layers. The improvements of client system performance justify additional complexity in a significant subset of deployment scenarios.

II. Related Work

There are two basic classes of sampling techniques: packet-based and flow-based methods. The *packet-based sampling* methods work on the level of the network packets. Each packet is selected for monitoring with a predefined probability depending on the sampling method used. The main advantage of the sampling deployment was the decreased requirements for memory consumption and CPU power on the routers as well as the possibility to monitor higher network speeds.

Although the packet sampling is easy to implement, it introduces a serious bias in flow statistics [4], [8]. The typical application of packet sampling on network traffic for traffic analysis, planning and management purposes has been studied in [5], [6]. Further research in packet sampling introduced adaptive packet sampling techniques [6], [2]. These techniques adjust the sampling rate depending on the current traffic load in order to acquire more accurate traffic statistics. An adaptive non-linear sampling method, which should preserve distribution of small flows and large flows is described in [9].

In case of *flow sampling*, the monitored traffic is aggregated into network flows and the sampling itself is applied not to the particular packets, but to the whole flows. The main benefit is better accuracy when compared to packet sampling [8], but they require more memory and CPU power.

Smart sampling [4] and sample-and-hold [7] techniques were introduces in order to reduce the memory requirements. Both of these techniques are focused on accurate traffic estimation for larger flows, so called *heavy-hitters*. The comparison of packet sampling with flow sampling is presented in [8]. The flow sampling is superior in flow distribution preservation, while the smart sampling prefers the large flows over the small ones. The comprehensive literature review can be found in [3].

Some recent papers do not focus only on the accuracy of sampling methods, but also on their on anomaly detection. The authors of [10] evaluate how the performance of anomaly detection algorithms is affected by random packet sampling, random flow sampling, smart sampling and sample-and-hold sampling. Their results demonstrate that the random packet sampling introduces a measurable bias and decreases the effectiveness of the detection algorithms. Overall, the random flow sampling proved to be a good choice.

The work by [1] proposes selective flow sampling, prioritizes small flows and therefore improves the results of several

Sampling Method	*Sampling Level*	*Volume Preserv.*	*Distrib. Preserv.*	*Anomaly Detection*
Random Packet Sampling	P	×	×	×
Adaptive Packet Sampling	P	○	×	×
Adaptive Non-Linear Sampling	P	●	×	×
Random Flow Sampling	F	●	○	○
Smart Sampling	F	●	×	×
Sample and Hold Sampling	F	●	×	×
Selective Sampling	F	×	×	●

TABLE I: The overview of the sampling methods, sampling levels (*P* - packet, *F* - flow) and their suitability for anomaly detection and for preserving traffic volumes and distributions. Legend: × - not suitable, ○ - partially suitable, ● - suitable.

anomaly detection methods. The presented results demonstrate that the sampling introduces serious bias in traffic feature distribution and therefore the effective use of this method is limited only to detection of specific types of anomalies.

The overview of the described sampling methods is presented in the Table I. We can see that the majority of sampling methods were designed for traffic monitoring, while the preservation of the traffic features crucial for the anomaly detection is suboptimal. We shall mention that the *random flow sampling method* provides relatively good results in all three areas. The *selective sampling method* represents the case of the sampling method directly designed for anomaly detection.

III. Model of Ideal Flow Sampling

In this Section, we introduce the ideal flow sampling process, which is used for selecting the most relevant flows (from the network security perspective). Intuitively, the ideal sampling should be a process in which number of samples and their distribution are selected in such a way that the loss of information is minimal. Almost all existing anomaly detection methods use statistical distribution of flows to model the network traffic. That means to minimize the loss of information, it is reasonable to preserve as much of the statistics as possible.

Each flow x can be identified by a set of features like source IP address or protocol. We will denote k-th feature as X_k and the probability of selecting a flow x into the sampled set as $p(x)$. Furthermore, the statistical information is captured by *feature moments* which are computed from feature values. We distinguish between two types of feature moments:

- *feature counts* $\mathsf{c}(x \mid X_k)$ indicating summations and numbers of flows related to x through the feature X_k (in flows, packets or bytes). We will implicitly use these feature counts in number of flows unless told otherwise.
- *feature entropies* $\mathsf{e}_{X_k}(x \mid X_l)$ describing the entropy of feature X_k of flows related to x through the feature X_l.

Finally we will denote the original finite unsampled set as U and the finite set of samples as S. Thus $\mathsf{c}^{\mathsf{S}}(x|srcIP)$ denotes the number of flows from the sampled set with exactly the same source IP as has this flow x. While $\mathsf{e}^{\mathsf{U}}_{sIP}(x|dP)$ is the entropy of source IP addresses from the original unsampled set, whose flows target exactly the same destination port as flow x. When we will consider feature counts across more (q) features, we will denote them as $\mathsf{c}(x|X_1,\ldots,X_q)$ etc.

Definition 1: Let $\mathsf{S}_1,\ldots,\mathsf{S}_m$ be various sets of flows selected from U with probability $p(x)$. Feature moment $\mathsf{c}(x|X_k)$ is reversible in U if and only if:

$$\forall x \in \mathsf{U} : \lim_{m\to\infty} \sum_{i=1}^{m} \left(\mathsf{c}^{\mathsf{U}}(x|X_k) \cdot p(x) - \mathsf{c}^{\mathsf{S}_i}(x|X_k) \right) = 0.$$

In the following, we will denote relative uncertainty RU, describing normalized entropy feature moment $\mathsf{e}_{X_k}(x|X_l)$, as:

$$RU(\mathsf{e}_{X_k}(x|X_l)) = \frac{\mathsf{e}_{X_k}(x|X_l)}{\log \mathsf{c}(x|X_l)} \in [0,1].$$

Definition 2: Let $\mathsf{S}_1,\ldots,\mathsf{S}_m$ be various sets of flows selected from U by using probability $p(x)$. Feature moment $\mathsf{e}_{X_k}(x|X_l)$ is reversible in U if and only if:

$$\forall x \in \mathsf{U} : \lim_{m\to\infty} \sum_{i=1}^{m} \left(RU(\mathsf{e}^{\mathsf{U}}_{X_k}(x|X_l)) - RU(\mathsf{e}^{\mathsf{S}_i}_{X_k}(x|X_l)) \right) = 0.$$

Definition 3: Let X_i be i-th flow feature. Feature variability $\mathbb{V}(X_i^{\mathsf{U}})$ of feature X_i is defined as the number of distinct values of feature X_i in U.

Definition 4: Ideal sampling with sampling probability $p(x)$ is defined as a sampling where:

1) all feature moments (counts and entropies) are reversible
2) coefficient $\frac{\mathbb{V}(X_i^{\mathsf{S}})}{\mathbb{V}(X_i^{\mathsf{U}})}$ for all features is maximized.

Each of the criteria caters to different kind of anomaly detection approaches: feature moment reversibility is essential for the methods based on statistical and pattern recognition methods, while the feature variability is also essential for knowledge-based approaches that depend on specific values of individual features. This idealistic process defines two actually usable quality metrics, that can be applied to any implemented sampling method in order to quantify the quality of the result it provides from the anomaly detection standpoint:

1) **feature representation** – describes the deviation (interval [0,1]) of a probability distribution from the ideal distribution, and thus measures the reconstruction error in the reversibility of count moment $\mathsf{c}(x|X_k)$:

$$f^{rep}_{\mathsf{c}}(X_k) = \frac{1}{\mid \mathsf{U} \mid} \cdot \sum_{\forall x \in \mathsf{U}} \mid \mathsf{c}^{\mathsf{S}}(x|X_k) - p(x) \cdot \mathsf{c}^{\mathsf{U}}(x|X_k) \mid$$

and the reconstruction error of entropy moment $\mathsf{e}_{X_k}(x|X_l)$:

$$f^{rep}_{\mathsf{e}}(X_k, X_l) = \frac{1}{\mid \mathsf{U} \mid} \cdot \sum_{\forall x \in \mathsf{U}} \mid RU(\mathsf{e}^{\mathsf{U}}_{X_k}(x|X_l)) - RU(\mathsf{e}^{\mathsf{S}}_{X_k}(x|X_l)) \mid$$

2) **feature coverage** – describes the variability of feature:

$$f^{cov}(X_i) = \frac{\mathbb{V}(X_i^{\mathsf{S}})}{\mathbb{V}(X_i^{\mathsf{U}})} \in [0,1].$$

We use these measures to compare the properties of proposed sampling technique (introduced in Section IV) with two existing sampling techniques (random and selective [1]).

IV. Approximation of the Ideal Flow Sampling

In this section, we will introduce an adaptive sampling algorithm that uses multi-stage processing and its knowledge of client anomaly detection algorithms to improve the sampling quality. The knowledge of features and moments used for anomaly detection in the subsequent stages allows us to reduce the sampling-induced errors where it matters the most.

A. Algorithm Structure and Late Sampling

The main challenge is the fact that the feature coverage and feature representation criteria are contradictory – improvement in one, while respecting an imposed sampling rate, directly negatively impacts the other. The algorithm we propose avoids making tradeoffs between these criteria by splitting the early stage of anomaly detection processing – feature extraction, and performing this operation **before** sampling. While this operation obviously breaks layering used in current systems, it is based on the rationale that the computational cost of the feature extraction phase is typically insignificant when compared to the cost of the rest of the processing.

The current techniques perform **early sampling**, where they compute the statistics (feature moments) after the sampling, so the moments suffer loss of precision [10]. The advantage of this, traditionally used sampling method is that there is no need to perform initial preprocessing of feature moments.

Using the **late sampling**, that we introduce in this paper, the system computes the moments *before* the sampling procedure, so the moment values are computed from the original, full set of data. This sampling method benefits from the fact that it can use exact statistical information about the original set, both for sampling technique itself, and for subsequent anomaly detection. However, there are specific scenarios, where this technique may not be applicable due to the nature of monitoring hardware or simply due to the extremely high traffic flow. The direct effect of late sampling is that it provides a perfect and unbiased information about the unsampled traffic.

B. Feature-Aware Sampling Algorithm

In this section, we will introduce the algorithm that optimizes the diversity of the measured flows by application of simple heuristics. *The algorithm is based on the assumption that the incremental value of flows in a single set (defined by one or more common feature values) decreases with the growing number of similar flows already in the set.* Therefore, the method is able to shrink the large sets of flows, while emphasizing the small artifacts that may be equally important from the security perspective. We assume that there are features with greater or lesser importance and impact on anomaly detection. We will denote the most important features as primary features and the rest is denoted as secondary features.

Definition 6: Let $X_1, \ldots, X_k$ be primary features. We define primary probability as the probability that a flow related to x through features $X_1, \ldots, X_k$ is selected to the sampled set:

$$p_p(x|X_1,\ldots,X_k) = \begin{cases} s & \mathsf{c}(x|X_1,\ldots,X_k) \le t \\ s \cdot \frac{\log t}{\log \mathsf{c}(x|X_1,\ldots,X_k)} & \mathsf{c}(x|X_1,\ldots,X_k) > t \end{cases}$$

where $s \in [0,1]$ is sampling rate and threshold t defines a point in the distribution, where our sampling technique starts to set the probability proportionally to the size of the moment. The higher the moment value, the lower sampling rate is assigned.

This modification from the random sampling slightly shifts the original probability distribution for flows with moment values above the threshold. We argue that reducing the size of attacks with higher values of moments (above the threshold) that are mostly easily detectable does not harm the anomaly detection effectiveness. Furthermore, such decrease in sampling rate allows to increase the sampling frequency when needed with no change in total amount of sampled flows.

Definition 7: Let $X_1, \ldots, X_k$ be primary features and let X_i be a secondary feature. We define secondary probability, which is a probability that a flow related to x through the feature X_i is selected to the sampled set, as:

$$p_s(x|X_i) = \begin{cases} d & \mathsf{c}(x|X_1,,X_k) > t \wedge RU(\mathsf{e}_{X_i}(x|X_1,,X_k)) \in I_1 \\ 1 & \text{otherwise} \end{cases}$$

$$I_1 \in [0,\varepsilon] \cup [1-\varepsilon, 1], \quad \varepsilon \in (0, 0.5),$$

where $d \in (0,1]$ is a parameter characterizing the decrease of incremental infomation value of the set with almost all identical flows ($RU \to 0$) or on the other side, mostly diverse flows ($RU \to 1$). Parameter ε determines the size of the interval, where the flows are considered as almost identical or mostly diversed.

Definition 8: Let $X_1, \ldots, X_k$ be primary features and $X_{k+1}, \ldots, X_n$ secondary features. Then the probability that the adaptive sampling will select flow x is defined as follows:

$$p(x) = p_p(x \mid X_1, \ldots, X_k) \cdot \prod_{i=k+1}^{n} p_s(x \mid X_i). \tag{1}$$

Theorem 1: Let $\mathsf{S}_1^{(r)}, \ldots, \mathsf{S}_m^{(r)}$ be sets of flows created by random sampling from U with sampling rate r. Let $\mathsf{S}_1^{(a)}, \ldots, \mathsf{S}_m^{(a)}$ be sets of flows created by the adaptive sampling with primary feature X_i and parameter s computed as:

$$s = \frac{r \cdot \mid \mathsf{U} \mid}{\sum_{\mathsf{c}(x|X_i) \le t} 1 + \sum_{\mathsf{c}(x|X_i) > t} \frac{\log t}{\log \mathsf{c}(x|X_i)}}. \tag{2}$$

Then it holds:

$$\overline{\mathsf{S}}^{(r)} = \lim_{m\to\infty} \left(\frac{1}{m} \cdot \sum_{i=1}^{m} |\mathsf{S}_i^{(r)}| \right) \ge \lim_{m\to\infty} \left(\frac{1}{m} \cdot \sum_{i=1}^{m} |\mathsf{S}_i^{(a)}| \right) = \overline{\mathsf{S}}^{(a)}.$$

Proof: When the random sampling technique is used, we can express the average number of sampled flows $\overline{\mathsf{S}}^{(r)}$ as follows:

$$\overline{\mathsf{S}}^{(r)} = r \cdot \mid \mathsf{U} \mid = s \cdot \left(\sum_{\mathsf{c}(x|X_i) \le t} 1 + \sum_{\mathsf{c}(x|X_i) > t} \frac{\log t}{\log \mathsf{c}(x|X_i)} \right)$$

Note that the summations sum all flows x satisfying the threshold condition. Now we can express $\overline{\mathsf{S}}^{(a)}$ as:

$$\overline{\mathsf{S}}^{(a)} = s \cdot \left(\sum_{\mathsf{c}(x|X_i) \le t} 1 + \sum_{\mathsf{c}(x|X_i) > t} \frac{\log t}{\log \mathsf{c}(x|X_i)} \right) - \varepsilon_p$$

where $\varepsilon_p \ge 0$ represents decrease in number of sampled flows caused by secondary probabilities. Computing parameter s according to the Eq. 2 guarantees the theorem statement. □

The proposed adaptive sampling is able to modify the sampling probability to reflect feature distributions of network

traffic. It selects flows according to the size of their moments in order both to suppress large, visible and easily detectable events, and to relieve some interesting facts from the smaller ones, while the feature distributions are slightly shifted for the benefit of the anomaly detection. Adaptive sampling also provides an upper bound in total number of sampled flows as stated in Theorem 1. This theorem guarantees that the total number of sampled flows does not exceed predefined limit.

V. Experimental Evaluation

The goal of the sampling evaluation is to compare various flow based sampling algorithms (we didn't focused on packet based sampling methods due to the significantly worse results [10]) at the real network traffic data. We performed two classes of evaluation. Firstly, we inspected the influence of the sampling methods on the traffic feature distributions and consequently, we evaluated the impact of sampling methods on the anomaly detection methods itself. The settings of the adaptive sampling was the following: we set $\mathsf{c}(x|srcIP)$ as the primary feature, $\mathsf{e}_{srcPrt}(x|srcIP)$, $\mathsf{e}_{dstIP}(x|srcIP)$ and $\mathsf{e}_{dstPrt}(x|srcIP)$ as the secondary features, $d = 0.8$, $\varepsilon = 0.1$ and $t = 1000$.

Because performing the experimental attacks directly in the real campus network (1Gb link) can harm the network services used by ordinary users, we decided to perform a set of experimental attacks in a separated testbed laboratory. These attacks were inserted to the live campus traffic background.

The first attack represents an escalated TCP vertical scan from one attacker IP address against one victim IP address. This attack started at 250 flows per 5 minutes only, but it grown up to 1 million flows progressively, in each successive 5 minute step. The second attack was motivated by the scenario, when the attacker launches a large DDoS attack, which is used for hiding the more serious SSH brute force attack (but with small intensity) against other victim in the same network.

A. Impact of Sampling on the Probability Distributions

In this Section, we will compare the differences of three sampling techniques (random, selective[4], and adaptive) from the ideal sampling by using measures described in Sec. III to evaluate the ability of preserving feature moments. For this evaluation, we used the network traffic from scenario with large DDoS and hidden SSH brute force attack.

First we evaluated two moments widely used for the anomaly detection - number of source IP addresses $\mathsf{c}(x|srcIP)$ and entropy of destination IP addresses for a given source IP address $\mathsf{e}_{dstIP}(x|srcIP)$. The lower the feature representation value, the more reversible the moment is. As you can notice from Table II, random sampling overcomes the other techniques in $\mathsf{c}(x|srcIP)$, while selective sampling confirms its selection speciality, which resulted in less reversible values. The adaptive sampling is shifted due to the variable sampling probability. However in case of the entropy moment, the adaptive sampling has the lowest reconstruction error.

In the second part of this evaluation, we compared feature coverage measure of source and destination IP addresses to discover how well each method preserves feature variability.

$f^{rep}(c_{srcIP}(x)),\quad f^{rep}(e_{dstIP}(x \mid srcIP))$			
rate	adaptive	random	selective
1:2	0.377, 0.284	0.205, 0.336	0.811, 0.414
1:5	0.500, 0.422	0.372, 0.444	0.946, 0.596
1:20	0.673, 0.600	0.600, 0.611	0.985, 0.749
1:100	0.936, 0.782	0.904, 0.783	0.996, 0.859

TABLE II: Feature representation measure for number of source IP addresses and entropy of destination IP addresses.

$f^{cov}(srcIP) \mid f^{cov}(dstIP)$			
rate	adaptive	random	selective
1:2	0.922 | 0.778	0.884 | 0.875	0.687 | 0.674
1:5	0.896 | 0.814	0.878 | 0.859	0.689 | 0.658
1:20	0.869 | 0.868	0.859 | 0.884	0.683 | 0.645
1:100	0.833 | 0.817	0.829 | 0.833	0.524 | 0.532

TABLE III: Feature coverage of source and dest. IP addresses.

As you can see from Table III, selecting source IP addresses as the primary feature of adaptive sampling clearly positively affected feature variability of source IP addresses, but negatively influenced variability of destination IP addresses.

In our evaluation, we clearly demonstrated that the moments of the adaptive sampling are much more reversible than the moments of selective sampling technique specialized on the anomaly detection. Some moments, especially those related to the primary features, are even more reversible than corresponding moments of random sampling, which makes the adaptive sampling a promising approach for preserving statistical information. Therefore the selection of the primary features is very crucial with respect to the detection techniques behind the sampling algorithm.

B. Impact of Sampling on the Anomaly Detection Methods

In this Section, we will present the evaluation of the detection quality of simulated attacks when using:

- unsampled and sampled data,
- various flow sampling techniques,
- statistical information based on sampled or full datasets (early and late sampling) – see Sec. IV-A.

More specifically, we compared the unsampled approach with four types of sampling techniques: Random E, Random L, Selective L and Adaptive L. The capitals E and L denotes early and late sampling described in Section IV-A. We compared the sampling methods according to both the number of selected flows from the attack and the detection quality, which is measured by using network behavioral anomaly device CAMNEP [11]. The detection quality represents the difference between the trustfulness of the global threshold ξ (which separates the flows into malicious or legitimate classes) and the average trustfulness of the attack flows $\overline{\Theta}(\varphi_j)$:

$$\text{Quality} = \xi - \overline{\Theta}(\varphi_j).$$

1) Scan Scenario: First we evaluated the sampling methods on the escalated TCP scan. In Fig. 1 (a) we compared each method quantitatively, i.e. according to the amount of selected traffic belonging to the simulated attack. The appropriateness of selective sampling method for scan detection is confirmed

by the increased numbers. Adaptive sampling selects smaller scans with higher probability than larger ones, which illustrates the shift on the left hand side. Larger (easily detectable) attacks were sampled with lower probability allowing the possibility of concentration on smaller events with no increase in the total amount of sampled traffic. This follows an intuition that the information value of individual flow in large attack is smaller.

The proportion of simulated TCP scan in sampled traffic is depicted in Fig. 1 (b). The final size of the attack occupies nearly 80% of sampled data when using selective sampling, while only 40% when using adaptive sampling. Thus the adaptive sampling has enough space in the sampled set for other interesting events on the network.

The quality of detection is depicted in Fig. 1 (c). Late selective, late adaptive and unsampled method successfully detected all sizes of the attack. On the contrary, late random and especially early random missed first smaller attacks.

2) Hidden SSH Brute Force Scenario: Next scenario contains a large DDoS attack, which covers more serious SSH brute force attack of smaller size (max 500 flows). Late selective sampling, which is not specialized on this type of attack, selected significantly less number of attack flows than late random and late adaptive, as illustrated in Fig. 1 (d). This Figure shows the proportion of the SSH brute force attack in sampled set depending on the size of escalated DDoS attack.

Graphical representation of the detection quality with sampling rate 1:5 can be seen in Fig. 1 (e). The worst and unsatisfactory results has early random sampling, while by using late sampling techniques the system detected the attacks more successfully (even better than unsampled approach), with late adaptive sampling slightly better than others. For selective sampling, the late approach has crucial effect on the detection quality, because it compensates the lack of attack flows in sampled set, so the system was able to recognize them.

When we decrease the sampling rate to 1:100, the situation becomes quite different (all methods failed), as you can see in Fig. 1 (f). Late selective sampling selected attack flows in only one dataset. The low sampling rate negatively influenced early random sampling as well as other late samplings.

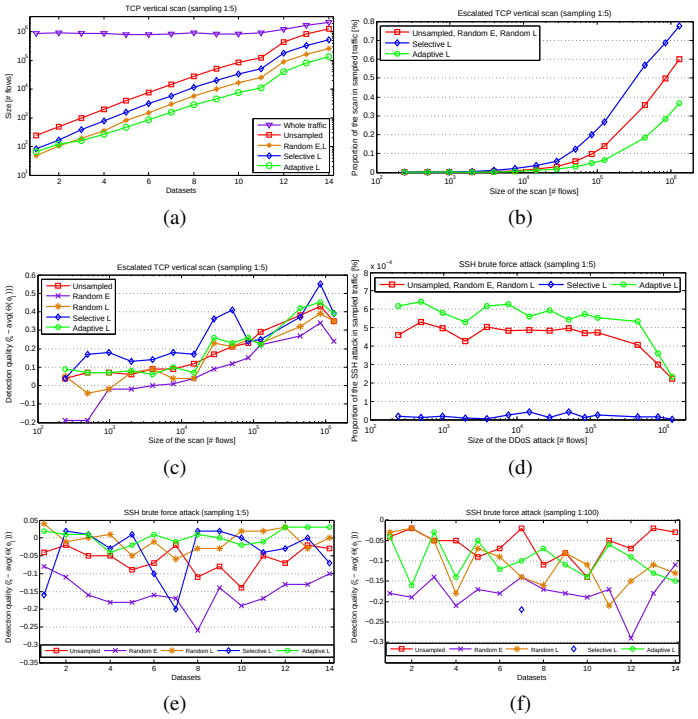


Fig. 1: Experimental evaluation.

VI. Conclusion

This paper has presented three principal contributions. First, we introduced formal model of ideal sampling together with two types of quality metrics, which we used to evaluate similarities between the ideal sampling and three other types of sampling algorithms. The metrics can be applied to any implemented sampling method in order to quantify the quality of the result it provides from the anomaly detection standpoint.

Next contribution of this paper was introducing a concept of late sampling, which significantly improved the performance of sampling methods. The late sampling technique provides exact statistical information about the original dataset, which makes this technique really suitable for anomaly detection environments that are based on those statistics.

Finally, we proposed the adaptive flow-based sampling method, which optimizes the sampling behavior with respect to the anomaly detection algorithms used later in the detection processing. The adaptive sampling is a promising general sampling technique that preserves well the traffic feature distributions and at the same time is able to improve the detection capabilities of the system.

References

[1] G. Androulidakis and S. Papavassiliou. Improving network anomaly detection via selective flow-based sampling. *Communications, IET*, 2(3):399–409, March 2008.

[2] B.-Y. Choi and Z.-L. Zhang. Adaptive random sampling for traffic volume measurement. *Telecommunication Systems*, 34(1-2):71–80, 2007.

[3] N. Duffield. Sampling for passive internet measurement: A review. *Statistical Science*, 19:472–498, 2004.

[4] N. Duffield, C. Lund, and M. Thorup. Properties and prediction of flow statistics from sampled packet streams. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 159–171, New York, NY, USA, 2002. ACM.

[5] N. Duffield, C. Lund, and M. Thorup. Estimating flow distributions from sampled flow statistics. *IEEE/ACM Trans. Netw.*, 13(5):933–946, 2005.

[6] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a better netflow. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 245–256, New York, NY, USA, 2004. ACM.

[7] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *ACM SIGCOMM Computer Communication Review*, volume 32, pages 323–336, New York, NY, USA, 2002. ACM.

[8] N. Hohn and D. Veitch. Inverting sampled traffic. *IEEE/ACM Trans. Netw.*, 14(1):68–80, 2006.

[9] C. Hu, S. Wang, J. Tian, B. Liu, Y. Cheng, and Y. Chen. Accurate and efficient traffic monitoring using adaptive non-linear sampling method. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 26–30, April 2008.

[10] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang. Is sampled data sufficient for anomaly detection? In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 165–176, New York, NY, USA, 2006. ACM.

[11] M. Rehak, M. Pechoucek, K. Bartos, M. Grill, P. Celeda, and V. Krmicek. CAMNEP: An intrusion detection system for high-speed networks. *Progress in Informatics*, (5):65–74, March 2008.